BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

As rescanning documents *will not* correct images, please do not report the images to the Image Problem Mailbox.

US006704278B1

(12) **United States Patent**
Albert et al.

(10) **Patent No.: US 6,704,278 B1**
(45) **Date of Patent: Mar. 9, 2004**

(54) **STATEFUL FAILOVER OF SERVICE MANAGERS**

(75) Inventors: **Mark Albert**, Wake Forest, NC (US); **Richard A. Howes**, Roswell, GA (US); **James A. Jordan**, Roswell, GA (US); **Edward A. Kersey**, Alpharetta, GA (US); **William M. LeBlanc**, Athens, GA (US); **Louis F. Menditto**, Raleigh, NC (US); **Chris O'Rourke**, Morrisville, NC (US); **Pranav Kumar Tiwari**, Raleigh, NC (US); **Bruce F. Wong**, Athens, GA (US)

(73) Assignee: **Cisco Technology, Inc.**, San Jose, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/347,123**

(22) Filed: **Jul. 2, 1999**

(51) **Int. Cl.**[7] **H04L 12/26**
(52) **U.S. Cl.** **370/216**
(58) **Field of Search** 370/216, 203, 370/230, 231, 236, 235, 349, 389, 390, 395.52, 428, 429, 400, 312, 410, 401; 709/224, 203; 714/36, 15, 23

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,774,660 A	*	6/1998	Brendel et al.	709/201
5,951,694 A		9/1999	Choquier et al.	714/15
6,006,264 A		12/1999	Colby et al.	709/226
6,128,642 A		10/2000	Doraswamy et al.	709/201
6,128,657 A		10/2000	Okanoya et al.	709/224
6,137,777 A		10/2000	Vaid et al.	370/230
6,148,410 A	*	11/2000	Baskey et al.	714/4
6,185,619 B1		2/2001	Joffe et al.	709/229
6,249,801 B1		6/2001	Zisapel et al.	709/105
6,263,368 B1		7/2001	Martin	709/224
6,327,622 B1		12/2001	Jindal et al.	709/228
6,330,602 B1		12/2001	Law et al.	709/224
6,366,558 B1	*	4/2002	Howes et al.	370/219
6,549,516 B1	*	4/2003	Albert et al.	370/236

OTHER PUBLICATIONS

Information Sciences Institute, "Internet Protocol, Darpa Internet Program Protocol Specification", Univ. of Southern Calif., Marina del Rey, CA. 90291, Sep. 1981.
S.Deering, "Host Extensions for IP Multicasting", Stanford University, Aug. 1989.

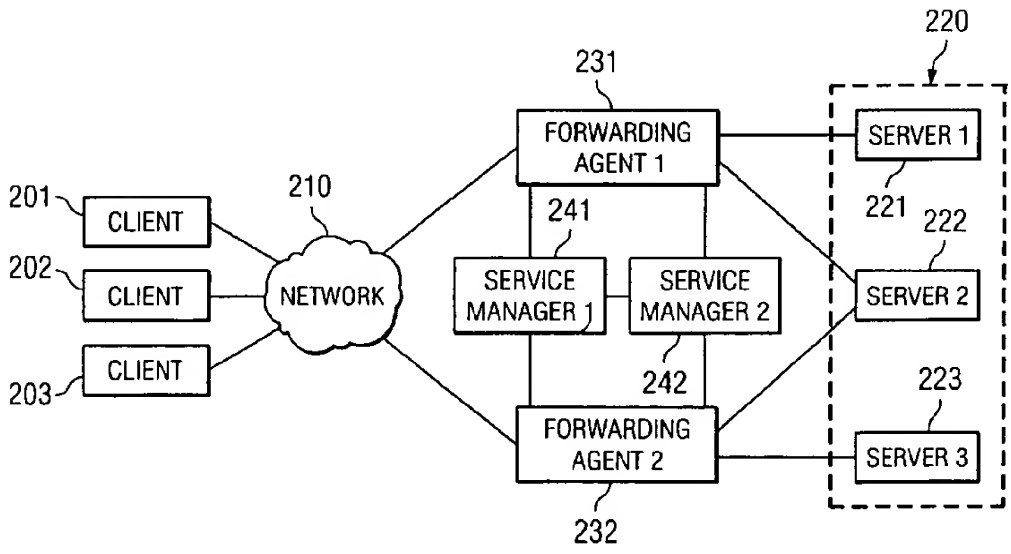
* cited by examiner

Primary Examiner—Chi Pham
Assistant Examiner—Alexander D. Boakye
(74) *Attorney, Agent, or Firm*—Baker Botts L.L.P.

(57) **ABSTRACT**

A system and method are disclosed for providing a fault tolerant network service. A packet is received that corresponds to a flow from a forwarding agent at a primary service manager and instructions are determined at the primary service manager for handling packets corresponding to the flow. The instructions are sent to the forwarding agent and the instructions are stored at the primary service manager. A replication packet is sent to a backup service manager. The replication packet includes the instructions for handling packets corresponding to the flow.

22 Claims, 8 Drawing Sheets



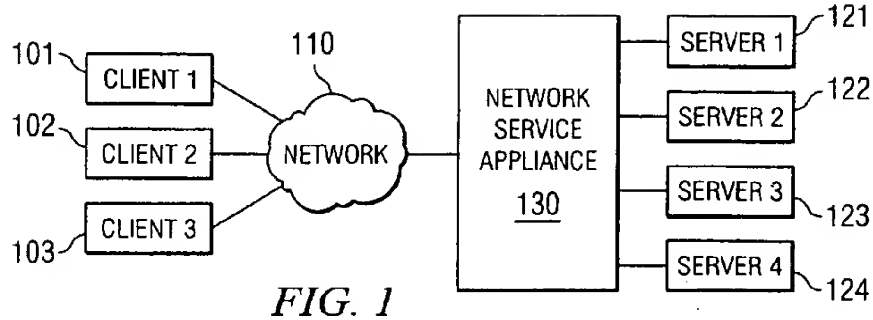


FIG. 1

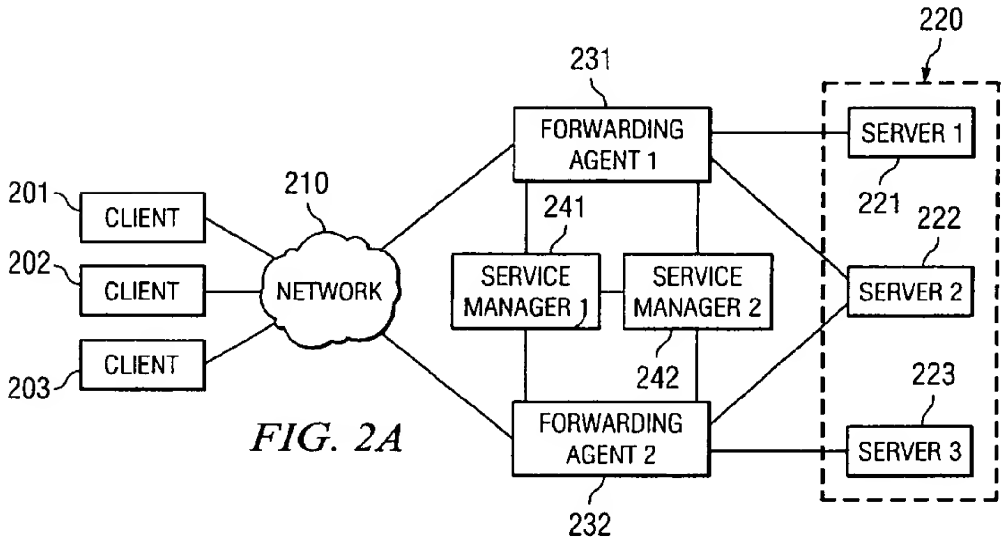


FIG. 2A

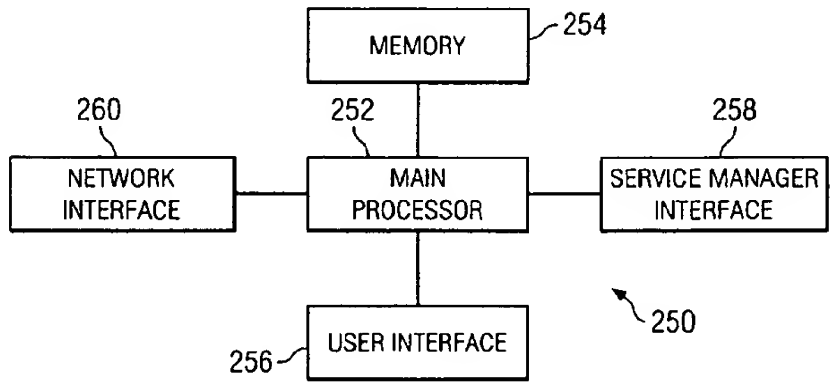


FIG. 2B

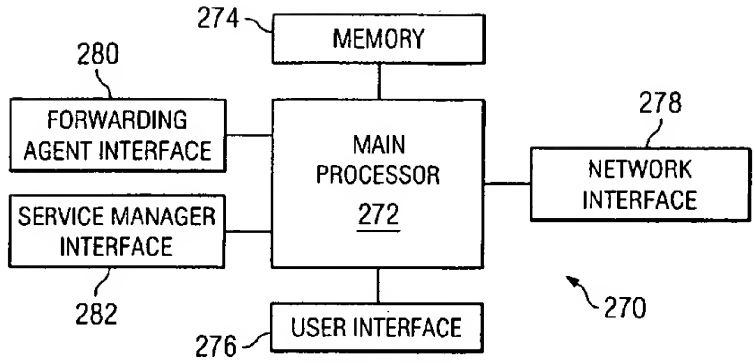


FIG. 2C

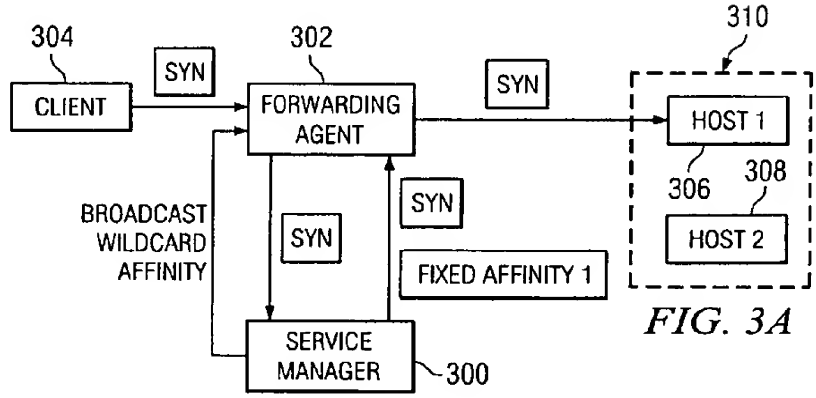


FIG. 3A

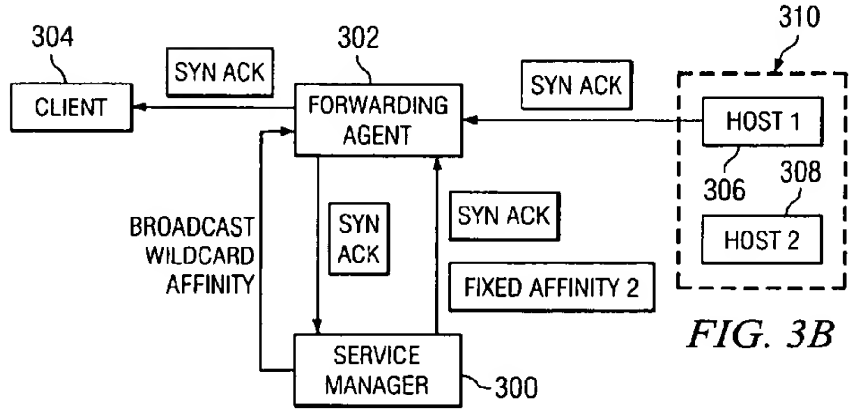


FIG. 3B

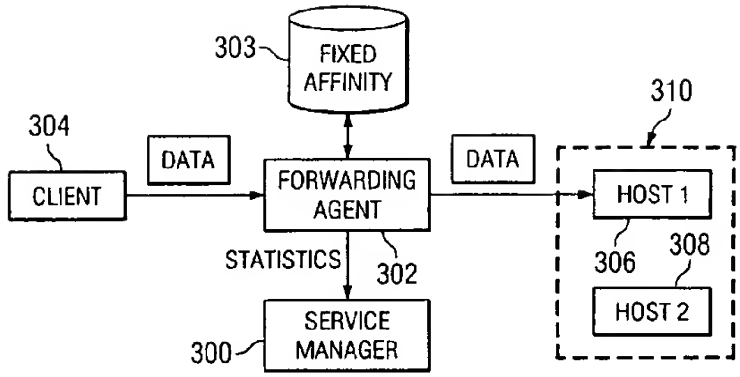

303
FIXED AFFINITY
310
304
DATA
DATA
CLIENT
FORWARDING AGENT
HOST 1
306
308
STATISTICS
302
HOST 2
SERVICE MANAGER
300

FIG. 3C

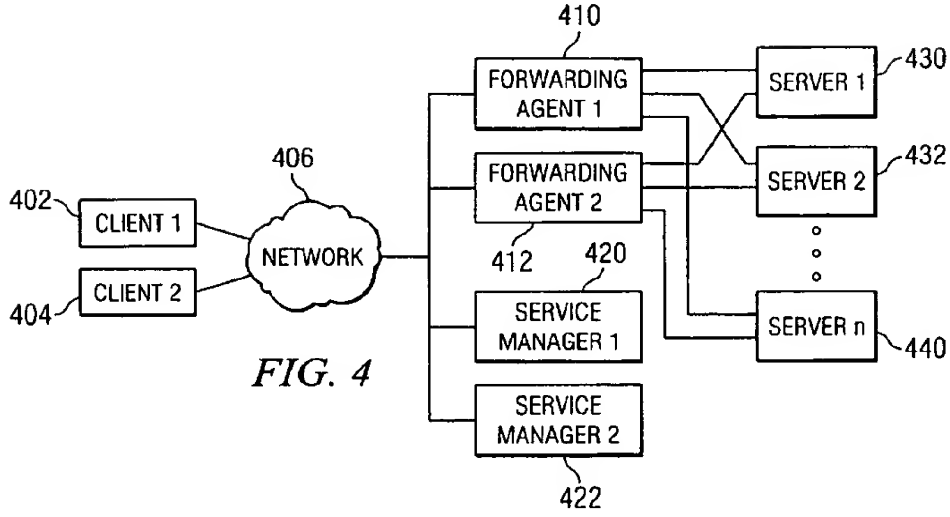

410
FORWARDING AGENT 1
SERVER 1
430
406
FORWARDING AGENT 2
SERVER 2
432
402
CLIENT 1
NETWORK
412
420
CLIENT 2
404
SERVICE MANAGER 1
SERVER n
440
SERVICE MANAGER 2
422

FIG. 4

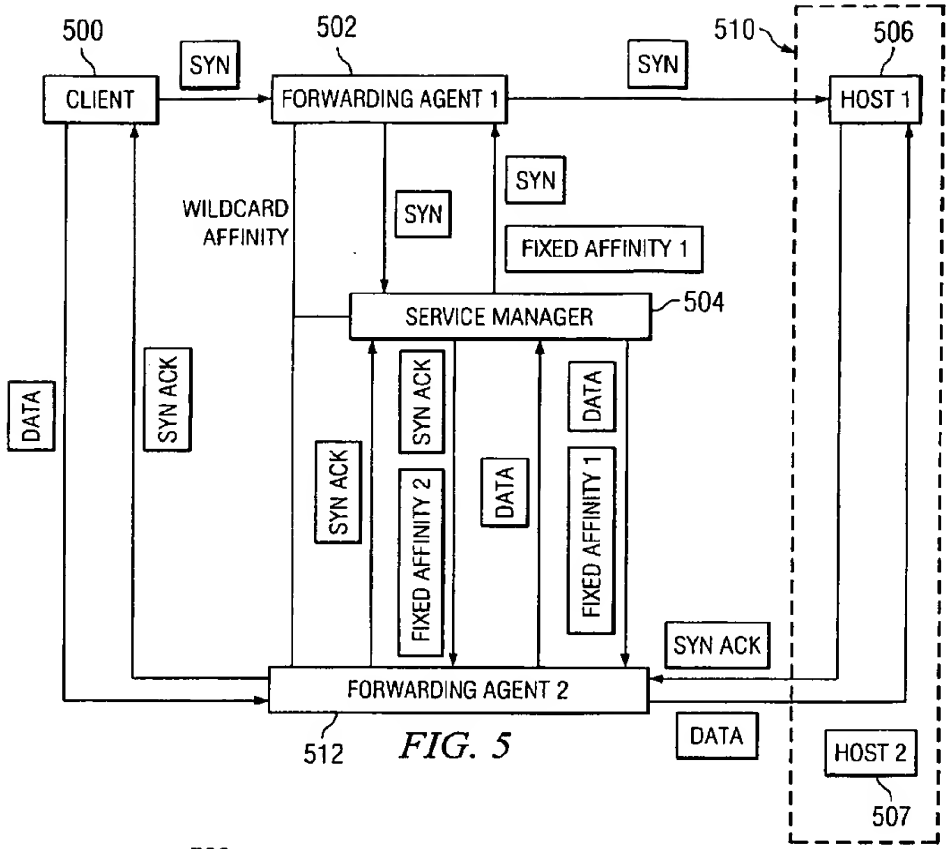


FIG. 5

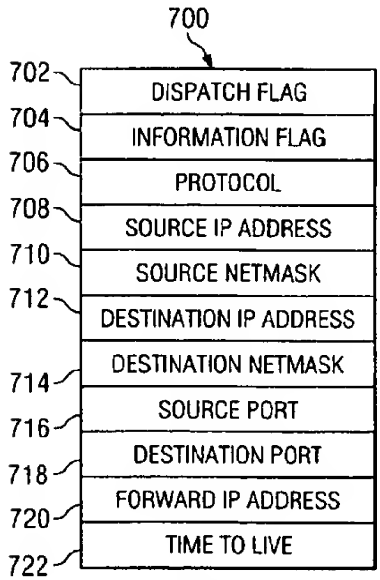


FIG. 7

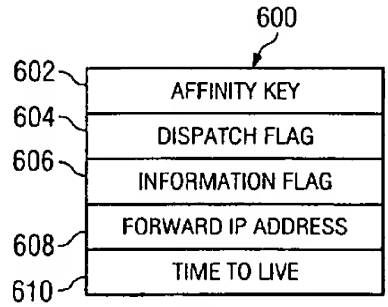


FIG. 6

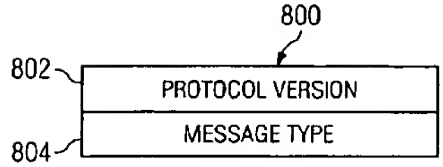


FIG. 8A

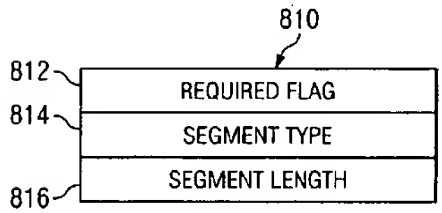


FIG. 8B

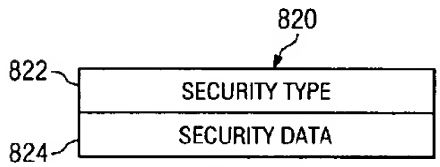


FIG. 8C

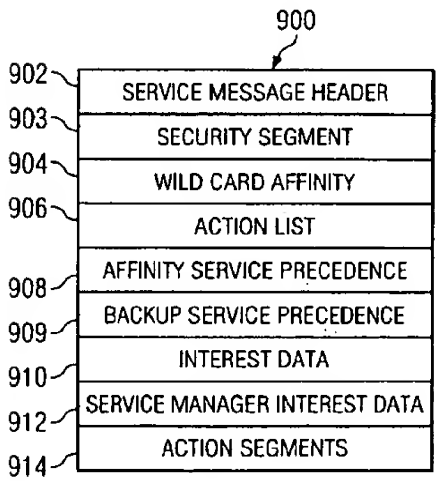


FIG. 9A

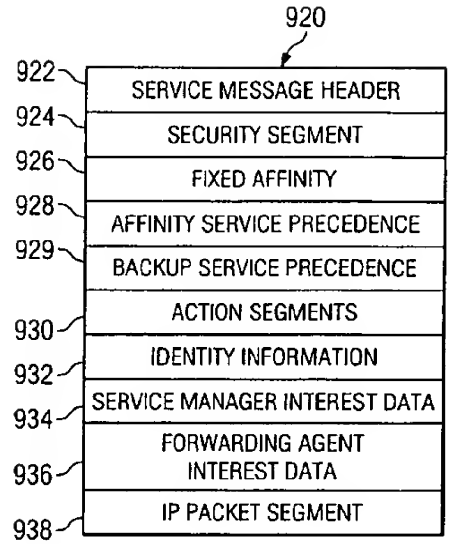


FIG. 9B

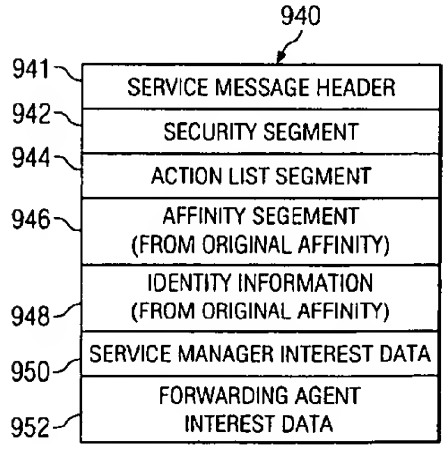


FIG. 9C

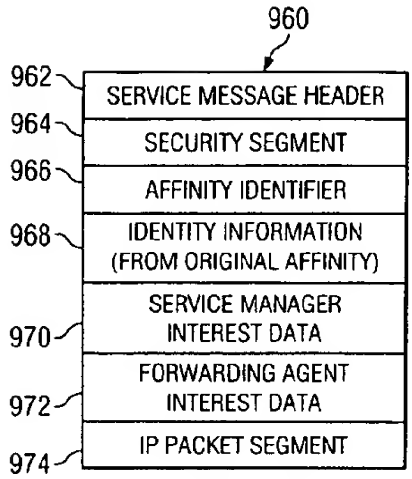


FIG. 9D

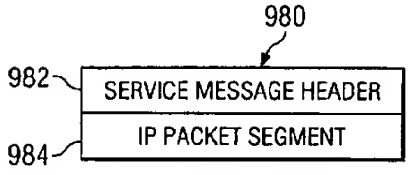


FIG. 9E

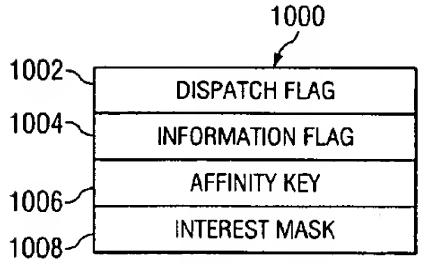


FIG. 10A

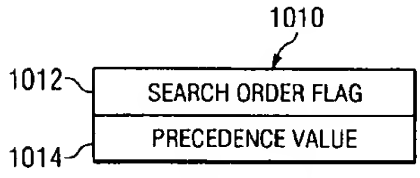


FIG. 10B

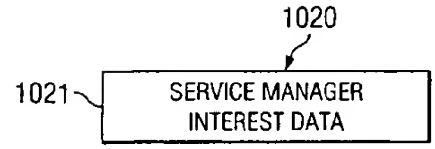


FIG. 10C

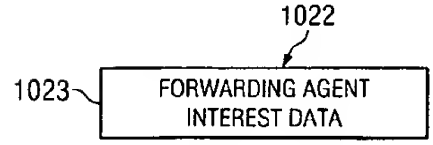


FIG. 10D

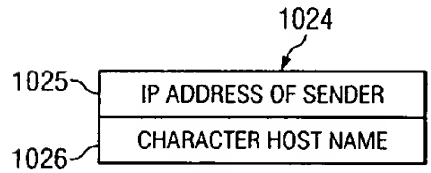


FIG. 10E

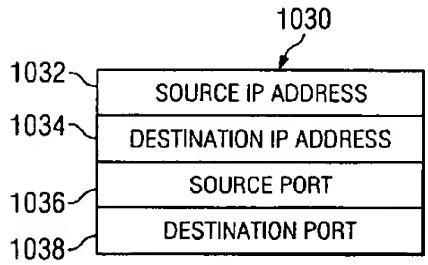


FIG. 10F

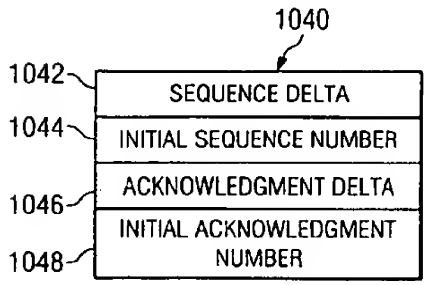


FIG. 10G

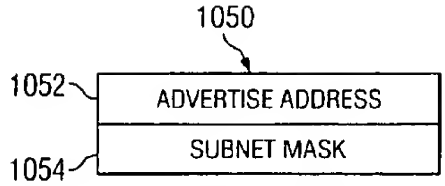


FIG. 10H

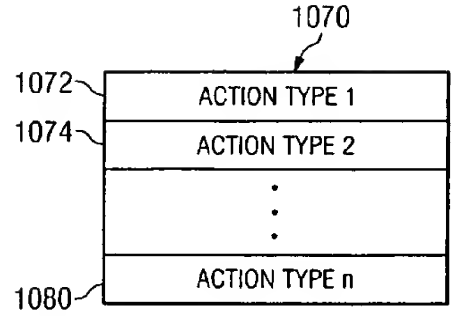


FIG. 10J

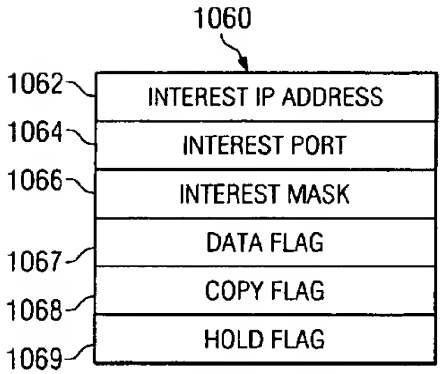


FIG. 10I

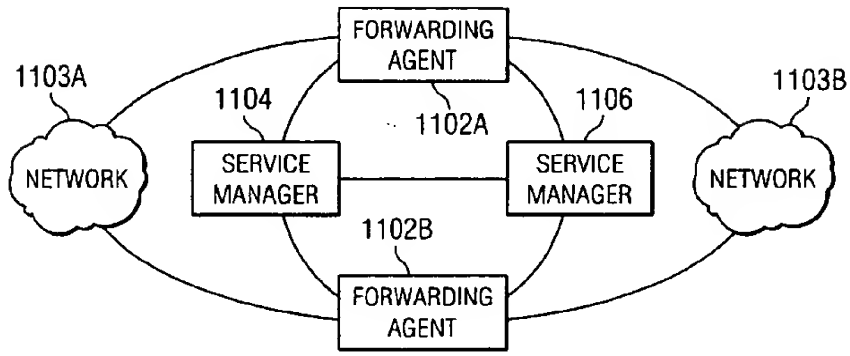


FIG. 11

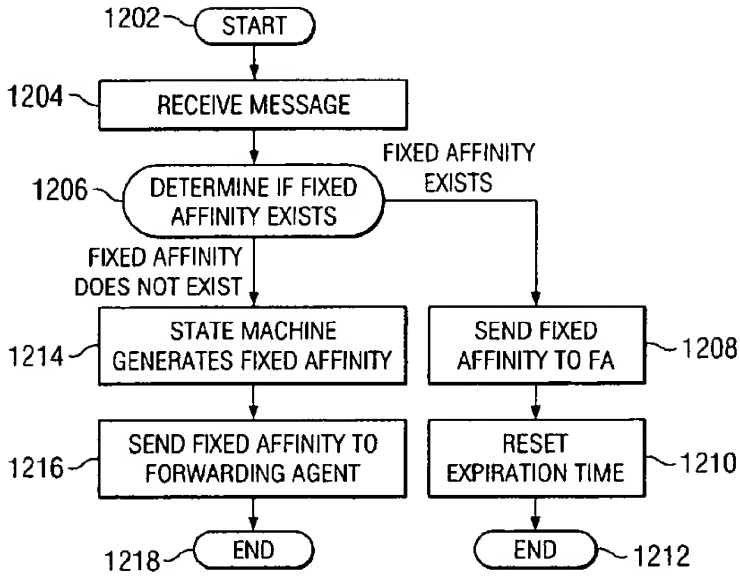


FIG. 12

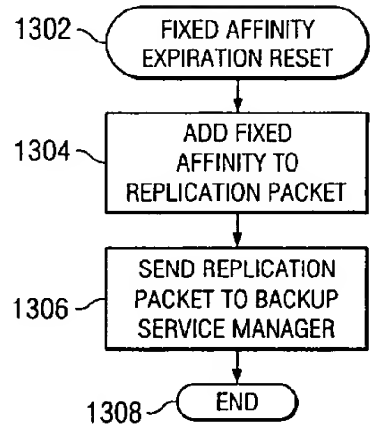


FIG. 13

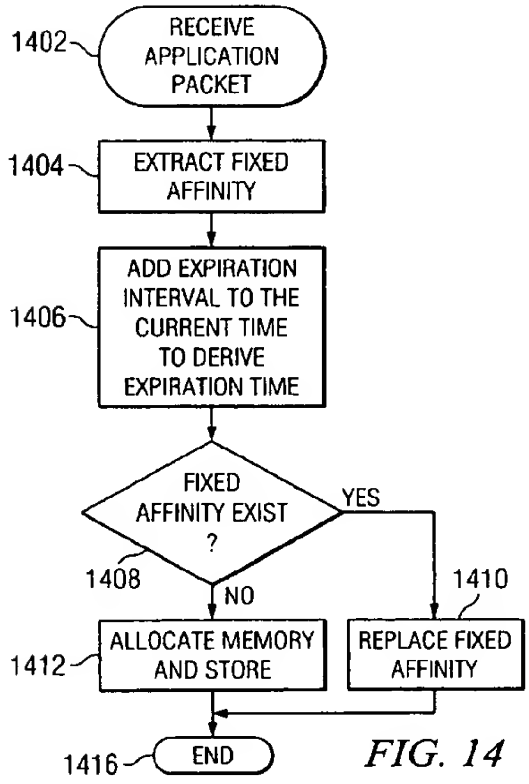


FIG. 14

STATEFUL FAILOVER OF SERVICE MANAGERS

CROSS REFERENCE TO RELATED APPLICATIONS

This application is related to U.S. patent application Ser. No. 09/346,634 now U.S. Pat. No. 6,628,654 entitled DISPATCHING PACKETS FROM A FORWARDING AGENT USING TAG SWITCHING; co-pending U.S. patent application Ser. No. 09/347,124 entitled CASCADING MULTIPLE SERVICES ON A FORWARDING AGENT; co-pending U.S. patent application Ser. No. 09/347,111 entitled LOAD BALANCING USING DISTRIBUTED FORWARDING AGENTS WITH APPLICATION BASED FEEDBACK FOR DIFFERENT VIRTUAL MACHINES; U.S. patent application Ser. No. 09/347,428 now U.S. Pat. No. 6,606,316 entitled GATHERING NETWORK STATISTICS IN A DISTRIBUTED NETWORK SERVICE ENVIRONMENT; co-pending U.S. patent application Ser. No. 09/347,122 entitled HANDLING PACKET FRAGMENTS IN A DISTRIBUTED NETWORK SERVICE ENVIRONMENT; U.S. patent application Ser. No. 09/347,108 now U.S. Pat. No. 6,549,516 entitled SENDING INSTRUCTIONS FROM A SERVICE MANAGER TO FORWARDING AGENTS ON A NEED TO KNOW BASIS; U.S. patent application Ser. No. 09/347,126 now U.S. Pat. No. 6,033,560 entitled DISTRIBUTION OF NETWORK SERVICES AMONG MULTIPLE SERVICE MANAGERS WITHOUT CLIENT INVOLVEMENT; co-pending U.S. patent application Ser. No. 09/347,034 entitled INTEGRATING SERVICE MANAGERS INTO A ROUTING INFRASTRUCTURE USING FORWARDING AGENTS; U.S. patent application Ser. No. 09/347,048 now U.S. Pat. No. 6,606,315 entitled SYNCHRONIZING SERVICE INSTRUCTIONS AMONG FORWARDING AGENTS USING A SERVICE MANAGER; co-pending U.S. patent application No. 09/347,125 entitled BACKUP SERVICE MANAGERS FOR PROVIDING RELIABLE NETWORK SERVICES IN A DISTRIBUTED ENVIRONMENT; co-pending U.S. patent application Ser. No. 09/347,109 entitled NETWORK ADDRESS TRANSLATION USING A FORWARDING AGENT; and co-pending U.S. patent application Ser. No. 09/347,036 entitled PROXYING AND UNPROXYING A CONNECTION USING A FORWARDING AGENT, all filed on Jul. 2, 1999 and incorporated herein by reference for all purposes.

FIELD OF THE INVENTION

The present invention relates generally to providing network services such as load balancing, packet filtering or Network Address Translation (NAT). More specifically, network services are provided using service managers and forwarding agents that are integrated into a routing infrastructure.

BACKGROUND OF THE INVENTION

As the IP protocol has continued to be in widespread use, a plethora of network service appliances have evolved for the purpose of providing certain network services not included in the protocol and therefore not provided by standard IP routers. Such services include NAT, statistics gathering, load balancing, proxying, intrusion detection, and numerous other security services. In general, such service appliances must be inserted in a network at a physical location where the appliance will intercept all flows of interest for the purpose of making its service available.

FIG. 1 is a block diagram illustrating a prior art system for providing a network service. A group of clients **101**, **102**, and **103** are connected by a network **110** to a group of servers **121**, **122**, **123**, and **124**. A network service appliance **130** is physically located in the path between the clients and the servers. Network service appliance **130** provides a service by filtering packets, sending packets to specific destinations, or, in some cases, modifying the contents of packets. An example of such modification would be modifying the packet header by changing the source or destination IP address and the source or destination port number.

Network service appliance **130** provides a network service such as load balancing, caching, or security services. In providing security services, network service appliance **130** may function as a proxy, a firewall, or an intrusion detection device. For purposes of this specification, a network service appliance that acts as a load balancer will be described in detail. It should be noted that the architecture and methods described are equally applicable to a network service appliance that is functioning as one of the other above described devices.

Network service appliance **130** is physically located between the group of servers and the clients that they serve. There are several disadvantages to this arrangement. First, it is difficult to add additional network service appliances when the first network service appliance becomes overloaded because the physical connections of the network must be rerouted. Likewise, it is difficult to replace the network service appliance with a back up network service appliance when it fails. Since all packets pass through the network service appliance on the way to the servers, the failure of the network service appliance may prevent any packets from reaching the servers and any packets from being sent by the servers. Such a single point of failure is undesirable. Furthermore, as networks and internetworks have become increasingly complex, multiple services may be required for a single network and inserting a large number of network service appliances into a network in places where they can intercept all relevant packet flows may be impractical.

The servers may also be referred to as hosts and the group of servers may also be referred to as a cluster of hosts. If the group of servers has a common IP address, that IP address may be referred to as a virtual IP address (VIPA) or a cluster address. Also, it should be noted that the terms client and server are used herein in a general sense to refer to devices that generally request information or services (clients) and devices that generally provide services or information (servers). In each example given it should be noted that the roles of client and server may be reversed if desired for a particular application.

A system that addresses the scalability issues that are faced by network service appliances (load balancers, firewalls, etc.) is needed. It would be useful to distribute functions that are traditionally performed by a single network element and so that as much function as possible can be performed by multiple network elements. A method of coordinating work between the distributed functions with a minimum of overhead is needed.

Although network service appliances have facilitated the development of scalable server architectures, the problem of scaling network service appliances themselves and distributing their functionality across multiple platforms has been largely ignored. Network service appliances traditionally have been implemented on a single platform that must be physically located at a specific point in the network for its service to be provided.

For example, clustering of servers has been practiced in this manner. Clustering has achieved scalability for servers. Traditional multiprocessor systems have relatively low scalability limits due to contention for shared memory and I/O. Clustered machines, on the other hand, can scale farther in that the workload for any particular user is bound to a particular machine and far less sharing is needed. Clustering has also facilitated non-disruptive growth. When workloads grow beyond the capacity of a single machine, the traditional approach is to replace it with a larger machine or, if possible, add additional processors within the machine. In either case, this requires downtime for the entire machine. With clustering, machines can be added to the cluster without disrupting work that is executing on the other machines. When the new machine comes online, new work can start to migrate to that machine, thus reducing the load on the pre-existing machines.

Clustering has also provided load balancing among servers. Spreading users across multiple independent systems can result in wasted capacity on some systems while others are overloaded. By employing load balancing within a cluster of systems the users are spread to available systems based on the load on each system. Clustering also has been used to enable systems to be continuously available. Individual application instances or machines can fail (or be taken down for maintenance) without shutting down service to end-users. Users on the failed system reconnect and should not be aware that they are using an alternate image. Users on the other systems are completely unaffected except for the additional load caused by services provided to some portion of the users that were formerly on the failed system.

In order to take full advantage of these features, the network access must likewise be scalable and highly available. Network service appliances (load-balancing appliances being one such example) must be able to function without introducing their own scaling limitations that would restrict the throughput of the cluster. A new method of providing network services using a distributed architecture is needed to achieve this.

In addition to being highly available, it would be useful if network services could be provided in a manner such that there is a smooth transition between a primary service manager and a backup service manager when the primary service manager fails. An efficient and reliable method of transferring state information between the primary service manager and the backup service manager is needed.

SUMMARY OF THE INVENTION

A system that includes a primary service manager and a backup service manager is disclosed. The primary service manager determines how a network service is provided and sends instructions to the forwarding agents that cause the forwarding agents to take appropriate actions. The primary service manager keeps track of flows that are being serviced and maintains instructions for the flows according to the traffic that the service manager monitors for the flows. In addition to sending instructions to the forwarding agents, the primary service manager also copies the instructions to the backup service manager, which maintains the instructions in parallel with the primary service manager. When the primary service manager fails, the backup service manager may begin servicing the flows formerly serviced by the primary service manager.

It should be appreciated that the present invention can be implemented in numerous ways, including as a process, an apparatus, a system, a device, a method, or a computer readable medium such as a computer readable storage medium or a computer network wherein program instructions are sent over optical or electronic communication links. Several inventive embodiments of the pre sent invention are described below.

In one embodiment, a fault tolerant method of providing a network service includes receiving a packet corresponding to a flow from a forwarding agent at a primary service manager and determining at the primary service manager instructions for handling packets corresponding to the flow. The instructions are sent to the forwarding agent and the instructions are stored at the primary service manager. A replication packet is sent to a backup service manager. The replication packet includes the instructions for handling packets corresponding to the flow.

In another embodiment, a primary service manager for providing a network service in a fault tolerant manner includes a processor configured to determine instructions for handling packets corresponding to a flow. A forwarding agent interface is configured to send the instructions for handling packets to a forwarding agent. A memory is configured to store the instructions for handling packets corresponding to the flow. A backup service manager interface is configured to send a replication packet to a backup service manager wherein the replication packet includes instructions for handling packets corresponding to the flow.

In another embodiment, a backup service manager for providing a network service in a fault tolerant manner includes a primary service manager interface configured to receive the instructions for handling packets corresponding to a now. A memory is configured to store the instructions for handling packets corresponding to the flow.

In another embodiment, a fault tolerant distributed system for providing a network service includes a forwarding agent configured to send a packet corresponding to a flow to a primary service manager. A primary service manager is configured to determine instructions for handling packets corresponding to the flow and to send the instructions for handling packets to the forwarding agent. The primary service manager stores the instructions for handling packets corresponding to the flow and sends a replication packet to a backup service manager. The replication packet includes the instructions for handling packets corresponding to the flow. A backup service manager is configured to receive the instructions for handling packets corresponding to the flow and to store the instructions for handling packets corresponding to the flow.

These and other features and advantages of the present invention will be presented in more detail in the following specification of the invention and the accompanying figures which illustrate by way of example the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

FIG. 1 is a block diagram illustrating a prior art system for providing a network service.

FIG. 2A is a block diagram of a network architecture that provides network services without requiring a network service appliance to be physically placed at a node through which all incoming and outgoing packets processed by a group of servers must pass.

FIG. 2B is a block diagram illustrating an architecture for a forwarding agent.

FIG. 2C is a block diagram illustrating an architecture for a service manager.

FIG. 3A is a diagram illustrating how a service manager and a forwarding agent cooperate to establish a connection from a client to a selected real machine.

FIG. 3B is a diagram illustrating how a forwarding agent routes a SYN ACK returned from a host back to a client.

FIG. 3C is a diagram illustrating how a subsequent data packet from client **304** is routed by forwarding agent **302** to host **306**.

FIG. **4** is a diagram illustrating a network that includes two forwarding agents and two service managers.

FIG. **5** is a diagram illustrating how a service manager provides instructions to two separate forwarding agents for handling a connection.

FIG. **6** is a diagram illustrating a fixed affinity.

FIG. **7** is a diagram illustrating a wildcard affinity.

FIG. **8**A is a diagram illustrating a service message header.

FIG. **8**B is a diagram illustrating a segment header.

FIG. **8**C is a diagram illustrating a security message segment.

FIG. **9**A is a diagram illustrating an affinity update wildcard message.

FIG. **9**B illustrates a fixed affinity update message that is sent by a service manager to a forwarding agent to add a fixed affinity to the receiver's affinity cache or delete a fixed affinity that is stored in the receiver's affinity cache.

FIG. **9**C is a diagram illustrating an affinity update-deny message.

FIG. **9**D is a diagram illustrating an interest match message for either a wildcard affinity or a fixed affinity.

FIG. **9**E is a diagram illustrating an IP packet only message.

FIG. **10**A is a diagram illustrating an affinity identifier segment.

FIG. **10**B is a diagram illustrating an affinity service precedence segment.

FIG. **10**C is a diagram illustrating a service manager interest data segment.

FIG. **10**D is a diagram illustrating a forwarding agent interest data segment.

FIG. **10**E is a diagram illustrating an identity information segment that is used to identify the sender of a service message.

FIG. **10**F is a diagram illustrating a NAT (Network Address Translation) action segment.

FIG. **10**G is a diagram illustrating a sequence number adjust action segment.

FIG. **10**H is a diagram illustrating an advertise action segment.

FIG. **10**I is a diagram illustrating an interest criteria action.

FIG. **10**J is a diagram illustrating an action list segment.

FIG. **11** is a block diagram illustrating a distributed network service architecture including service managers and forwarding agents.

FIG. **12** is a flow chart illustrating a process executed by a service manager for managing fixed affinities.

FIG. **13** is a flowchart illustrating the process for sending a replication packet to the backup service manager.

FIG. **14** is a flowchart illustrating a process implemented on the backup service manager upon the receipt of a replication packet.

DETAILED DESCRIPTION

A detailed description of a preferred embodiment of the invention is provided below. While the invention is described in conjunction with that preferred embodiment, it should be understood that the invention is not limited to any one embodiment. On the contrary, the scope of the invention is limited only by the appended claims and the invention encompasses numerous alternatives, modifications and equivalents. For the purpose of example, numerous specific details are set forth in the following description in order to provide a thorough understanding of the present invention. The present invention may be practiced according to the claims without some or all of these specific details. For the purpose of clarity, details relating to technical material that is known in the technical fields related to the invention has not been described in detail in order not to unnecessarily obscure the present invention in such detail.

FIG. **2**A is a block diagram of a network architecture that provides network services without requiring a network service appliance to be physically placed at a node through which all incoming and outgoing packets processed by a group of servers must pass. Several clients **201**, **202**, and **203** are connected to a network **210**. Network **210** is connected to a group of servers **220** that includes servers **221**, **222**, and **223**. There is no point through which all traffic between devices connected to network **210** and the group of servers **220** must pass. Instead, some traffic from network **210** that is bound for the group of servers passes through a forwarding agent **231** and some traffic between network **210** and group of servers **220** passes though a forwarding agent **232**.

In the example shown, forwarding agent **231** is connected to server **221** and server **222** and forwarding agent **232** is connected to server **222** and server **223**. Thus, server **222** may communicate with network **210** through either of the forwarding agents, server **221** communicates with network **210** exclusively through forwarding agent **231**, and server **223** communicates with network **210** exclusively through forwarding agent **232**. This arrangement may be generalized to include an arbitrary number of servers connected to an arbitrary number of forwarding agents with individual servers connected to arbitrary subsets of the forwarding agents.

A service manager **241** and a second service manager **242** also communicate with the forwarding agents. The service managers provide the decision making capability that is required to provide a network service such as load balancing. The service managers send specific instructions to each of the forwarding agents detailing how certain flows of packets are to be processed. Such packet processing may include simply routing the packet, gathering statistics about the packet, sending the packet to a service manager, sending a notification that the packet has been seen to a service manager, modifying the packet, or using a special method such as tunneling or tag switching to send the packet to a destination other than the destination specified by the destination IP address included in the packet header. It should also be noted that forwarding agents in other embodiments also modify other aspects of packets, including packet source and destination addresses and port numbers and, in some instances, packet data.

The service managers communicate with the forwarding agents to give the agents instructions relating to how to handle packets for various flows that are routed through the forwarding agents. It is useful at this point to review certain terminology used herein relating to connections and flows.

As used in this specification, a connection consists of a set of flows. A flow is a set of related packets sent between two

end stations. A flow may be identified with layer 3 and layer 4 parameters, depending on the protocol being used. For example, for TCP and UDP, a flow is identified by five parameters: the source and destination IP addresses and port numbers and the protocol. For ICMP, flows are defined by three parameters: the source and destination IP addresses and the protocol.

TCP connections will be described in detail in this specification. It should be appreciated that the techniques disclosed apply to other types of connections as well. TCP connections are defined by a 5-tuple that includes the source and destination IP addresses, the source and destination port numbers, and an identification of the protocol that applies to the packet. The source and destination IP addresses and ports for packets going in one direction between the devices are reversed for packets going in the opposite direction. That is, when the direction that a packet is travelling is reversed, the source becomes the destination and the destination becomes the source. Packets flowing in one direction of a connection are in the same flow.

A connection transfers data between applications on two machines having IP addresses and the applications correspond to port numbers. If the protocol is set by convention to be a certain protocol such as TCP, then a protocol identifier may not be required. The 4 remaining numbers, the source and destination IP addresses, and the source and destination port numbers, are sometimes referred to as a quad. In this specification, the 5-tuple that includes the source and destination IP addresses, the source and destination port numbers and a protocol identification will be referred to as an affinity key. Each unique affinity key thus defines a flow in one direction of a connection. If the source and destination IP addresses and port numbers are reversed for a single affinity key, then it becomes an affinity key that corresponds to a flow in the opposite direction for the same connection. In general, a flow may be identified by a source IP address and destination IP address, by a source IP address, destination IP address and protocol, by a quad, by an affinity key 5-tuple, by only a source and destination IP address or by other information available in a packet header. The term, "flow identifier" is intended to refer to any such method of identifying a flow.

Affinity keys are used by the service managers to identify flows passing through forwarding agents which are to be handled by the forwarding agents in a certain manner. Forwarding agents can accomplish their required tasks with only limited processing capability. Forwarding agents need not determine how to handle certain flows or make decisions such as load balancing or security decisions relating to the flows. The service manager performs those functions and forwards specific instructions to forwarding agents detailing exactly what actions are to be taken for each flow. Instructions for how to handle packets are specified for each flow by the service managers using an affinity key. A specific affinity key that is sent to a forwarding agent together with instructions detailing how packets for flows specified by the affinity key are to be handled is referred to as a fixed affinity.

In addition to specifying instructions for each flow, service managers must also obtain information about each new flow from the forwarding agents. For example, when a service manager provides load balancing through a set of forwarding agents, the service manager uses fixed affinities to provide specific instructions to the forwarding agents detailing where packets for each load balanced flow are to be forwarded. In addition to providing those specific instructions, the service manager also provides general instructions to each forwarding agent that specify which new flows the service manager is interested in seeing. These general instructions are provided using wildcard affinities. Wildcard affinities, which are described in detail below, specify sets of flows that are of interest to a service manager. In one embodiment, this is done by specifying subnet masks that determine sets of source and destination IP addresses that will be forwarded to a service manager. In addition, ports or sets of ports and protocol may be specified in wildcard affinity as well. As is described further below, the use of wildcard affinities enables separate service managers to be configured to provide services for different sets of flows. Each service manager specifies the flows of interest to it and other service managers handle other flows. In this manner, service managers can be configured in parallel to share load.

Thus, service managers use wildcard affinities to specify flows for which they may be providing service and forwarding agents transfer packets for new flows to the appropriate service manager. Once a service manager determines how a certain flow is to be handled, the service manager sends a fixed affinity to each forwarding agent. The fixed affinity overrides the wildcard affinity stored in the forwarding agent that instructs the forwarding agent to forward packets to the service manager with specific instructions for the specific flow specified by an affinity key in the fixed affinity.

In the case of load balancing, service managers send wildcard affinities to forwarding agents. The wildcard affinities specify destination IP addresses that correspond to virtual IP addresses of server clusters that are to be load balanced by the service manager. The forwarding agents then forward new packets sent to those virtual IP addresses to the appropriate service manager. The service manager selects a server from the server cluster and then the service manager sends a fixed affinity to each forwarding agent that instructs the forwarding agent to forward packets for that specific flow to the selected server in the cluster. Forwarding agents may also forward packets for purposes other than load balancing. Packets may be forwarded to real IP addresses as well as virtual IP addresses.

In one embodiment, each forwarding agent is implemented on a router. In other embodiments, forwarding agents may be implemented on switches or other network devices and may be implemented on a coprocessor in a device that also performs another network function. When implemented on a router, the power of this architecture becomes clear. By infusing each router with a limited functionality provided by the forwarding agent, the service managers are able to provide network services without physically being inserted at the various points in the network where those services must be provided. The physical presence of each of the routers at those points is sufficient to enable network services to be provided. This contradicts the conventional wisdom regarding the restriction that all traffic inbound for a server cluster must pass through a single load-balancing engine. The combination of fast forwarding agents (be they 'routers' or IP-aware 'switches') and service managers (to provide synchronization and control) eliminates the scalability limitations of the past.

This specification will refer in detail to forwarding agents implemented on routers for the purpose of example. It should be remembered that forwarding agents may also be implemented on other devices and that the same or similar advantages may be realized.

The service managers send wildcard affinities to each of the forwarding agents that direct the forwarding agents to process packets that match the wildcard affinities in a certain

manner. For example, a service manager may request to be notified when certain packets are received by the routers that include the forwarding agents. When a packet that matches such an instruction is received, the forwarding agent notifies the service manager and the service manager determines what to do with that packet and future packets for the flow based on the network service being provided. Instructions are then sent from the service manager to the forwarding agent at the router that allow the router to process the packets in accordance with the decisions made by the service manager.

In addition to specifying that a service manager is to be notified upon receipt of a certain type of packet, wildcard affinities may also specify other actions to be taken. For example, a wildcard may specify an IP address to which packets are to be forwarded without notification to the service manager. Packets may also be copied to a service manager or other device and packets may also be denied or dropped.

It should be noted that the service managers also may be connected to one or more of the servers and may in some cases forward packets received from forwarding agents or received from the network directly to certain servers. However, it is significant that the service managers need not be connected to servers for which they are managing packet traffic. The service manager may accomplish all packet routing through forwarding agents by sending instructions to forwarding agents. It should also be noted that the service managers may also be connected to each other for the purpose of coordinating their instructions or providing backup services.

FIG. **2B** is a block diagram illustrating an architecture for a forwarding agent. Forwarding agent **250** includes a main processor **252** and a memory **254**. Memory **254** may include RAM, ROM, nonvolatile memory such as an EPROM, or a disk drive. Forwarding agent **250** also includes a user interface **256** that allows a user to configure the forwarding agent or monitor the operation of the forwarding agent.

Forwarding agent **250** also includes a service manager interface **258** that allows packets to be sent to and received from a service manager. In addition, the service manager interface allows service managers to send fixed and wildcard affinities to the forwarding agent. In one embodiment, a separate interface is used for the purpose of sending wildcard affinities to forwarding agents using multicast. In other embodiments, a single interface may be provided between the service manger and the forwarding agent. The forwarding agent also includes a network interface **260** that is used to send and receive packets to and from other devices on the network.

It should be noted that the network interface and the service manager interface may be the same interface in certain embodiments. In such embodiments, all communication between the forwarding agent and the service manager is carried on the same network as packets processed by the forwarding agent.

A forwarding agent may be implemented on various network devices. A forwarding agent may be implemented on a network device dedicated to acting as a forwarding agent but the true power of the system is realized when forwarding agents are implemented on network devices that already are included in a network for some other purpose. Forwarding agents may be implemented on routers that already exist at strategic points in a network for intercepting packets and providing a service using a forwarding agent.

FIG. **2C** is a block diagram illustrating an architecture for a service manager. Service manager **270** includes a main processor **272** and a memory **274**. Memory **274** may include RAM, ROM, nonvolatile memory such as an EEPROM or a disk drive. Service manager **270** also includes a user interface **276** for the purpose of allowing a user to configure the service manager or monitor the operation of the service manager.

Service manager **270** also optionally includes a network interface **278**. Network interface **278** allows the service manager to directly forward packets into the network for which it is providing a service. If no network interface is provided, then the service manager can still forward packets by sending them to a forwarding agent.

A forwarding agent interface **280** is included on the service manager for the purpose of allowing the service manager to send packets and affinities to forwarding agents. Forwarding agent interface **280** may include more than one interface. For example, in one embodiment, a separate interface is used for multicasting wildcard affinities to all forwarding agents and a separate interface is used for the purpose of unicasting fixed affinities to individual forwarding agents and forwarding packets to individual forwarding agents.

Service manager **270** may also include a service manager interface **282** used to communicate with other service managers. The service manager may communicate with other service managers for the purpose of providing a fail over scheme of backup service managers. Operational status of service managers may be communicated on the service manager interface and a master service manager may send configuration information about flows being supported through backup service managers so that the backup service managers can function in place of the master service manager should it fail.

A service manager may be implemented on a standard microcomputer or minicomputer. In one embodiment a service manager is implemented on a UNIX workstation. A Service manager may also be implemented on other platforms including Windows, an embedded system or as a system on a chip architecture. A service manager also may be implemented on a router.

One network service that can be readily provided using the architecture described in FIG. **2A** is load balancing connections among a set of real machines that are used to service connections made to a virtual machine. The real machines may also be referred to as hosts and the virtual machine may also be referred to as a cluster of hosts. The following figures describe how a service manager directs forwarding agents to intercept packets for new connections and send them to the service manager. The service manager then selects a real machine to handle each connection, and directs one or more forwarding agents to forward packets to the selected real machine. Forwarding agents may forward packets using NAT or may use another method of sending packets to the selected real machine.

FIG. **3A** is a diagram illustrating how a service manager and a forwarding agent cooperate to establish a connection from a client to a selected real machine. A service manager **300** broadcasts or multicasts a wildcard affinity to all forwarding agents that are listening for wildcard affinities sent by service manager **300**. In some embodiments, wildcard affinities may be broadcast. A forwarding agent **302** receives the wildcard affinity. In one embodiment, all forwarding agents and service managers register to a common multicast group so that neither service managers nor forwarding agents need to have any preknowledge of the existence of each other. Thus, a service manager registers its interests

with the forwarding agents by multicasting wildcard affinities to the multicast group. Each wildcard affinity provides a filter which recognizes general classes of packets that are of interest.

As an example, client **304** may wish to establish a TCP connection with a virtual machine having a virtual IP address. It should be noted that other types of connections may also be established. To establish the TCP connection, client **304** sends a SYN packet with a destination address corresponding to the virtual IP address. The SYN packet is received by forwarding agent **302**. Forwarding agent **302** determines that the destination address of the SYN packet matches the wildcard affinity broadcast by service manager **300**. The action included in the broadcast wildcard affinity specifies that all packets matching the wildcard affinity are to be forwarded to the service manager. Therefore, forwarding agent **302** forwards the SYN packet to service manager **300**.

Service manager **300** receives the SYN packet from the forwarding agent. It should be noted that, in one embodiment, forwarding agent **302** encapsulates the SYN packet in a special system packet when the SYN packet is sent to the service manager. Service manager **300** receives the SYN packet and processes the packet according to whatever service or services are being provided by the service manager. In the example shown, service manager **300** is providing load balancing between a first host **306** and a second host **308**. Together, host **306** and host **308** comprise a virtual machine that services the virtual IP address that is the destination of the SYN packet sent by client **304**. Service manager **300** determines the host that is to receive the SYN packet and that is to handle the connection initiated by the SYN packet. This information is included in a fixed affinity. The SYN packet is encapsulated with the fixed affinity and sent back to forwarding agent **302**.

The fixed affinity sent to the forwarding agent **302** may include an action that directs the forwarding agent to dispatch the SYN packet directly to host **306**. The action included in the fixed affinity may also direct the forwarding agent to translate the destination address of the packet to the IP address of host **306** and the packet may be routed to host **306** via one or more hops. In addition, as described below, tag switching may also be used to send the packet to the host that is selected by the service manager using its load balancing algorithm.

Thus, the SYN packet is directed to the host selected by service manager **300** without service manager **300** being inserted into the path of the packet between the hosts which comprise virtual machine **310** and client **304**. The service manager broadcasts a wildcard affinity to all forwarding agents potentially in that path and the forwarding agents forward SYN packets to the service manager whenever a client establishes a new connection. The service manager then returns the SYN packet with a fixed affinity that directs the forwarding agent how to forward that SYN packet as well as future packets sent in the flow from the client to the virtual machine. The forwarding agent then sends the SYN packet on to the selected host using network address translation (NAT), tag switching, or some other method.

FIG. **3B** is a diagram illustrating how a forwarding agent routes a SYN ACK returned from a host back to a client. A service manager **300** broadcasts a wildcard affinity to a forwarding agent **302**. The wildcard affinity matches packets with a source IP address matching either host **306** or host **308** which implement virtual machine **300**. When host **306** sends a SYN ACK packet back to client **304**, the SYN ACK travels through forwarding agent **302**. Because of the wildcard affinity that matches the source IP address of host **306**, forwarding agent **302** encapsulates the SYN ACK packet and sends it to service manager **300**. Service manager **300** then identifies the SYN ACK as the SYN ACK corresponding to the SYN that was sent by the client shown in FIG. **3A** and sends the SYN ACK together with a fixed affinity to forwarding agent **302**. The fixed affinity may include an action that directs the forwarding agent to replace the source IP address of host **306** with the virtual IP address of virtual machine **310** before forwarding the SYN ACK packet on to client **304**.

Thus, FIGS. **3A** and **3B** show how a forwarding agent intercepts a SYN packet from a client and translates the destination IP address from the destination IP address of a virtual machine to the destination IP address of a specific host. The specific host is determined by the service manager using a load balancing algorithm. The forwarding agent does not include logic that performs load balancing to determine the best host. The forwarding agent only needs to check whether the incoming SYN packet matches a fixed affinity or a wildcard affinity broadcast to the forwarding agent by the service manager.

The SYN packet is forwarded to the service manager and the service manager returns the SYN packet to the forwarding agent along with a fixed affinity that includes an action which specifies how the forwarding agent is to handle the SYN packet. When a SYN ACK is returned by the host, the forwarding agent again finds a wildcard affinity match and forwards the SYN ACK packet to the service manager. The service manager returns the SYN ACK packet to the forwarding agent along with a second fixed affinity that instructs the forwarding agent how to handle packets in the flow back from the host the client.

The first fixed affinity from the service manager includes an affinity key that corresponds to the flow from the client to the host and the second fixed affinity sent form the service manager to the forwarding agent contains an affinity key that corresponds to the flow from the host back to the client. Future packets in either flow sent from the client or the host match the affinity key in one of the fixed affinities and are handled by the forwarding agent according to the action contained in the fixed affinity. It is no longer necessary to forward such packets to the service manager. In some applications, the forwarding agent may continue to forward data about the packets to the service manager so that the service manager can monitor connections or maintain statistics about network traffic.

FIG. **3C** is a diagram illustrating how a subsequent data packet from client **304** is routed by forwarding agent **302** to host **306**. Client **304** sends a data packet to forwarding agent **302**. Forwarding agent **302** has stored the fixed affinity corresponding to the flow from the client to the host in a fixed affinity database **303**. Forwarding agent **302** notes the match of the 5-tuple of the data packet with an affinity key in the fixed affinity database and then forwards the data packet according to the action defined in that fixed affinity. In this example, the action defined is to translate the destination IP address of the client from the virtual IP address of virtual machine **310** to the IP address of host **306**. In addition to forwarding the data packet, the affinity found by the forwarding agent also includes an action that requires the forwarding agent to send an affinity packet to service manager **300** that includes data about the packet for the purpose of service manager **300** gathering statistics about network traffic.

The examples shown in FIG. **3A** through FIG. **3C** illustrate how the first packet sent in both flows of a new

connection are forwarded to the service manager by the forwarding agent. The service manager then directs the forwarding agent to handle the packets in a certain manner by sending fixed affinities to the forwarding agent for each flow and specifying actions to be performed on the packets. In the example shown, the action involves translating the destination IP address from the client to a specific host IP address and translating the source IP address in packets form the host to a virtual IP address. Other actions may be defined by fixed affinities including translating other IP addresses, translating port numbers or dispatching packets to other machines. Some of these other actions are described below.

FIG. **4** is a diagram illustrating a network that includes two forwarding agents and two service managers. A first client **402** and a second client **404** send packets through a network or internetwork **406** that eventually reach a subnetwork that includes a first forwarding agent **410**, a second forwarding agent **412**, a first service manager **420**, and a second service manager **422**. In the examples shown, the service managers communicate with the forwarding agents and with each other over the same physical network that is used to send packets. In other embodiments, a separate physical connection may be provided between service managers for the purpose of coordinating service managers and providing back up service managers and a separate connection may be provided between the service managers and the forwarding agents for the purpose of multicasting wildcard affinities or, in some embodiments, for sending fixed affinities and returning packets to forwarding agents.

In general, the service managers may communicate amongst themselves and with the forwarding agents in any manner appropriate for a particular system. The forwarding agents each are connected to a first server **430**, a second server **432** and other servers up to an nth server **440**. These servers may represent one or more virtual machines. Packets from the clients may be routed through either forwarding agent **410** or forwarding agent **412**. In fact, packets corresponding to the same connection or flow may be routed at different times through different forwarding agents. To cope with this situation, the service managers multicast wildcard affinities to both forwarding agents. When either forwarding agent first receives a packet for a flow, that forwarding agent forwards the packet to the manager that has requested the packet using a wildcard affinity so that the service manager can provide the forwarding agent with the fixed affinity that defines how to handle the packet.

FIG. **5** is a diagram illustrating how a service manager provides instructions to two separate forwarding agents for handling a connection. A client **500** sends a SYN packet to a first forwarding agent **502**. Forwarding agent **502** has previously received a wildcard affinity from a service manager **504** on a dedicated connection on which service manager **504** multicasts wildcard affinities to forwarding agents. As a result of the wildcard match, forwarding agent **502** encapsulates the SYN packet and forwards it to service manager **504**. Service manager **504** receives the SYN packet and returns it to forwarding agent **502** along with a fixed affinity specifying an action to be performed on the packet. The action defined in this example is translating the destination IP address of the packet from a virtual IP address to the IP address of a host **506**. Hosts **506** and **507** together implement a virtual machine **510**.

Host **1** receives the SYN packet from forwarding agent **1** and returns a SYN ACK packet back to client **500**. However, for some reason, the SYN ACK packet from host **1** is routed not through forwarding agent **502**, but instead through forwarding agent **512**. Forwarding agent **512** receives the SYN ACK and notes that it matches a wildcard affinity corresponding to the flow of packets from host **506** to client **500**. Forwarding agent **512** encapsulates the SYN ACK packet and sends it to service manager **504**. Service manager **504** defines an action for the SYN ACK packet and includes that action in a second fixed affinity which it sends along with the encapsulated SYN ACK packet back to forwarding agent **512**. Forwarding agent **512** then sends the SYN ACK packet on to client **500** where it is processed.

At this point, forwarding agent **502** has a fixed affinity for the flow from client **500** to the hosts and forwarding agent **512** has a fixed affinity for the flow from the hosts back to client **500**. Each forwarding agent continues to handle flows without fixed affinities using the wildcard affinities. The service manager acts as a point of synchronization between the forwarding agents when the forwarding agents handle common flows.

Client **500** then sends a data packet which happens to be routed through forwarding agent **512** and not forwarding agent **502**. Forwarding agent **502** has received the fixed affinity that provides instructions on how to deal with packets in the flow from client **500** to virtual machine **510**. However, forwarding agent **512** has not yet received that fixed affinity. Forwarding agent **512** has received a wildcard affinity previously multicast by the service manager. Therefore, forwarding agent **512** detects a wildcard affinity match for the data packet and encapsulates the data packet and sends it to service manager **504**.

Service manager **504** receives the data packet and notes that the data packet matches the previously defined first fixed affinity which was sent to forwarding agent **502**. Service manager therefore does not run the load balancing algorithm again to determine where to route the data packet, but instead returns the first fixed affinity to forwarding agent **512** along with the data packet. Forwarding agent **512** receives the data packet and the fixed affinity and then has the same instructions as forwarding agent **502** for handling that data packet and other packets in the flow from client **500** to virtual machine **510**. Forwarding agent **512** therefore translates the destination IP address of the data packet to the IP address of host **506** and forwards the packet on to host **506**.

Thus, as long as wildcard affinities are received by each forwarding agent, the service manager is able to provide fixed affinities to each forward agent whenever a fixed affinity is required to provide instructions to handle packets for a given flow. Once a fixed affinity is defined for a flow, the same fixed affinity is provided to any forwarding agent that returns a packet to the service manager as a result of a wildcard match.

To provide a load balancing service for HTTP, a service manager sends a pair of wildcard affinities (one for each direction of flow to and from a virtual machine) to a multicast group that includes each available router in a network. The wildcard affinities specify a protocol and also indicate an exact match on the IP Address and HTTP port number for the virtual machine and an IP address and mask combination that identifies the client population that is serviced by the service manager. The client population serviced by the service manager is referred to as the client domain of the service manager. If multiple service managers are used, then each service manager may be configured to service a different client domain.

For example, if the majority of traffic is coming from a small number of firewalls, whereby the same foreign IP address is shared by many different clients, all those affini-

ties can be assigned by one service manager. Thus, traffic from large sites can be isolated from other traffic and assigned to a different service manager.

Thus, the architecture is scalable and service managers may be added to handle client domains as needed. The set of clients serviced by each service manager can be changed by canceling the wildcards that each service manager has broadcast to forwarding agents and sending new wildcards specifying the new client domain.

When multiple service managers are included, it is important that the client domains specified by service managers performing the same service do not overlap. The task of assigning affinities for each client domain is centralized by the service manager serving that domain so all packets for a given flow are controlled by a single service manager. For example, if duplicate SYN packets are sent by a client, both should be directed to the same service manager and assigned the same fixed affinity. If the packets were directed to different service managers, then the service manager load balancing algorithms might assign different real machines to handle the connections as a result of the network being in a different state when the second SYN packet arrived. In addition, UDP unicasts from the same client must be assigned the same affinity and related connections (e.g., FTP control and data connections) must be assigned the same affinity.

Once the forwarding agents have received fixed affinities, packets intercepted that match a fixed affinity are processed as instructed in the set of actions specified in the fixed affinity. If a matching fixed affinity is not found, the packet is compared against the wildcard affinities to find manager (s) that are interested in this type of packet. If no appropriate Wildcard Affinity is found, normal IP routing occurs. Generally, a manager uses the wildcard affinity to be informed of flows it may be interested in. Once a manager has determined how a flow should be handled, it usually sends a fixed affinity so that the processing of subsequent packets.for that flow can be offloaded to the forwarding agent. In some cases actions for certain flows can be predetermined by the service manager without seeing packets from the flow. In such cases, the actions may be specified in a wildcard affinity and no message need be sent to the service manager and no fixed affinity need be generated. The service manager may specify that it is still to receive certain packet types after a fixed affinity is sent by including an optional action interest criteria message segment with the fixed affinity.

In the load-balancing case, a fixed affinity is used to identify the server that is to receive this particular flow whereas a wildcard affinity is used to define the general class of packets for which load balancing is to be performed (all those matching the cluster address and port number for the clustered service) and to identify the manager that is to make the balancing decision for flows that match the wildcard affinity.

Fixed Affinities

FIG. 6 is a diagram illustrating a fixed affinity **600**. Fixed affinity **600** matches only one flow through a network. As described above, a flow is defined by an affinity key, which is a unique 5-tuple that spans the packet headers:

IP Header:

Protocol Type (e.g., UDP or TCP)

Source IP Address

Destination IP Address

TCP or UDP Header:

Source Port

Destination Port

It should be noted that if the protocol being used is not TCP or UDP, then the ports in the affinity key may be set to 0.

Fixed affinity **600** includes an affinity key **602**. In addition, fixed affinity **600** contains information that dictates how a forwarding agent is to process packets that match the affinity key, and how the forwarding agent is to manage the affinity.

A dispatch flag **604** indicates whether the packet is to be dispatched to the forward IP address included in the fixed affinity. Setting the dispatch flag indicates that the packet is to be forwarded to a forward IP address **608** that is provided in the fixed affinity. The difference between dispatched and directed traffic is that dispatch traffic is forwarded directly from a forwarding agent to a specific server without translating the destination IP address of the packet. In other words, if a packet is dispatched, then the packet destination address is not used to forward the packet. Instead, a forwarding address contained in an affinity is used to forward the packet. If the connection is not dispatched but directed by the forwarding agent, then the packet IP destination must be translated using NAT if the packet is redirected to a specific server.

If forward IP address **608** is zero, then the packet is dropped after processing statistics as indicated by an information flag **606**. Not setting the dispatch flag indicates that the packet is to be forwarded based on the address provided in the packet IP header.

Information flag **606** indicates whether or not statistics are to be gathered for packets forwarded using the fixed affinity. If the Information flag is set, statistics are updated for the forward IP address. In one embodiment, the statistics kept include:

1. total bytes for all packets matching t he forward P address
2. total packets matching the forward P address

Statistics for packets and bytes matching the affinity may be kept regardless of the setting of the Information flag.

Fixed affinity **600** also includes a time to live **610**. Time to live **610** specifies the number of seconds before the fixed affinity should be timed-out from a fixed affinity cache maintained by a forwarding agent. If a time to live of 0 is specified, then that means that the fixed affinity is not to be cached by a forwarding agent and if a copy of the fixed affinity is already in the cache, it should be removed. Thus, service managers may remove fixed affinities that they have sent to forwarding agents by simply sending copies of those fixed affinities to the forwarding agents with time to live set to 0.

Each fixed affinity sent by a service manager is correlated to a wildcard affinity previously sent by the service manager. If a forwarding agent receives a fixed affinity for which no supporting wildcard affinity is found, the forwarding agent ignores the fixed affinity and discards it.

Wildcard Affinities

FIG. **7** is a diagram illustrating a wildcard affinity **700**. Wildcard affinity **700** is a more general form of Affinity that is used by a service manager to register filters with the forwarding agent(s) that define the range of flows that are of interest to the service manager. Like a fixed affinity, wildcard affinity **700** also includes a dispatch flag **702** and an information flag **704**. Wildcard affinity **700** also includes the elements of an affinity key (protocol **706**, source IP address **708**, destination IP address **712**, source port **716**, and destination port **718**) plus source netmask **710** and destination netmask **714**.

The netmasks and the source and destination IP addresses are used to specify ranges of addresses covered by the wildcard affinity. The source netmask is ANDed with the source IP address in the wildcard affinity. The source netmask is also ANDed with the source IP address from the packet. If the results of the two operations are equal, then the source IP address of the packet is considered to be in range of the wildcard affinity. Likewise, the destination netmask is ANDed with the destination IP address in the wildcard affinity. The destination netmask is also ANDed with the destination IP address from the packet. If the results of the two operations are equal, then the destination IP address of the packet is considered to be in range of the wildcard affinity. If both the source and the destination IP addresses of the packet are in the range of the wildcard affinity, and the ports and protocols also match, then the packet is said to match the wildcard affinity. It should also be noted that, in one embodiment, a zero specified for a port or a protocol matches all ports or protocols.

It should be noted that in other embodiments, other methods of specifying ranges for the wildcard affinity are used. For example, in one alternative arrangement, ranges of IP addresses are specified by specifying lower bound and upper bound IP addressees. All addresses between the two bounds fall within the range of the wildcard affinity. In some applications, multiple ranges may be specified. The method described above is particularly useful for specifying a single address, specifying all addresses in a subnet, or specifying every even or odd address, every fourth address, every eighth address, etc.

For example, to specify a single host of 1.1.1.1, the wildcard affinity include an IP address of 1.1.1.1 with a netmask of 255.255.255.255. To specify the range of hosts from 1.1.1.0 to 1.1.1.255, the wildcard affinity would include an IP address of 1.1.1.0 with a netmask of 255.255.255.0, indicating that the first three bytes of the IP address must match exactly and that the last byte is to be ignored.

Wildcard affinity **700** also includes a time to live **722**. Time to live **772** is used in the same manner as the time to live for the fixed affinity. Wildcard affinities are deleted by forwarding agents based on the time to live set for the wildcard affinity by the service manager. The timing of such a deletion need not be exact. In one embodiment, the timing need only be accurate to within two seconds. This same tolerance is for fixed affinities as well. Service managers must refresh each wildcard affinity before its time to live expires in order to continue to receive packets that match the wildcard affinity from forwarding agents. As with the fixed affinity, a wildcard affinity may be deleted by sending a duplicate wildcard affinity with a time to live of 0.

Actions

Thus, fixed affinities specify individual flows and packets and wildcard affinities specify sets of flows to be processed in a special way. Such processing is defined by associating actions with the affinities. Actions defined for the affinities specify the service to be performed by the forwarding agent on behalf of the Manager. For fixed affinities, services specified may include:

- Interest Criteria—a list of packet types that cause a notification to be sent to the service manager.
- Sequence Number Adjustment—a set of deltas and initial sequence numbers by which the TCP sequence numbers and ACK numbers are to be adjusted.
- NAT—provides details for how Network Address Translation is to be performed.

For Wildcard Affinities, applicable actions are:

- Interest Criteria—a list of packet types that cause a notification to be sent to the service manager.
- Advertise—indicates that the destination IP Address in the Wildcard Affinity is to be advertised by the forwarding agent. This may be done by including the destination IP address in routing protocol updates.
- Sequence Number Adjustment—a set of deltas and initial sequence numbers by which the TCP sequence numbers and ACK numbers are to be adjusted.
- NAT—provides details for how Network Address Translation is to be performed.

Forwarding agents may not support all possible actions. For example, some forwarding agents may not support NAT. The set of actions that the service manager expects a forwarding agent to support are identified in an action list which may be included with the wildcard affinity. If the forwarding agent does not support one or more of the actions identified in the list, it discards the wildcard affinity and send a message to the service manager indicating that it does not support all of the actions in the list. This message is referred to as an affinity update deny message. The service manager then may attempt to send a new wildcard affinity that excludes any unsupported actions identified in the affinity update deny message.

Service Messages

Wildcard affinities, fixed affinities, actions, packets, and other messages are sent between service managers and forwarding agents encapsulated in service messages. In one embodiment, messages sent between service managers and forwarding agents are sent using the specific service message format described below. Service messages are sent between service managers and forwarding agents using UDP. Wildcard affinities, which are sent by service managers, can be multicast to a multicast IP Address and UDP Port known to the service manager(s) and forwarding agent(s), or can be unicast to a particular forwarding agent or service manager. FIG. **8**A is a diagram illustrating a service message header used in one embodiment. Service message header **800** includes a protocol version **802** and a message type **804**. The protocol version identifies the version of the service protocol supported by the sender. The message type identifies the overall purpose of this message, the base format for the message, and implies the set of optional message segments that may be included in the message.

The following service message types are used:

Message Type
affinity update-wildcard affinity
affinity update-fixed affinity
affinity update-deny
interest match-wildcard affinity
interest match-fixed affinity
IP packet only

The affinity update-wildcard affinity message is used to send wildcard affinities from a service manager to forwarding agents. The affinity update-fixed affinity message is used to send fixed affinities. The affinity update-deny message is used to report that an affinity update message has been rejected because required actions included in the affinity update are not supported by the receiver. The interest match-wildcard affinity message is used to report a wildcard affinity match to a service manager and the interest match-

fixed affinity message is used to report a fixed affinity match to a service manager. The IP packet only message is used to forward an IP packet.

After the service message header, a service message includes one or more message segments. Each message segment begins with its own segment header. FIG. **8B** is a diagram illustrating a segment header. Segment header **810** includes a Required flag **812**. Required flag **812** defines whether the sender will allow the rest of the message to be processed even if the segment cannot be processed (either because the receiver does not support the function described by the segment or because the receiver does not understand the segment). The required flag either indicates that the segment may be ignored or that the segment is required. If a required segment cannot be processed, then the entire message that includes the segment is dropped and an error message is returned to the sender. Each segment header is followed by data that is specific to the message segment.

The following message segments are used:

Segment Name
Wildcard Affinity
Fixed affinity
Affinity Interest
Service Precedence
Security
Service Manager Interest Data
forwarding agent Interest Data
Identity Info
Action-NAT
Action-Advertise
Action-Sequence Number Adjust
Action-Interest Criteria
Action List
IP Packet

The fixed affinity, wildcard affinity and security segments are described immediately below. The remaining segments are described in detail following a description of the message types that include the segments.

Security

If security is expected by the receiver, a security message segment immediately follows the service message header. The security message segment contains the expected security sequence. If the receiver does not expect security, the security message segment is ignored (if present) and the message is accepted. Security is generally not required for IP packet only messages. If authentication is successful, the signals are accepted. If the authentication fails, the signal is ignored. Various authentication schemes such as MD5 may be supported. The type of authentication to be used is configured at the senders and receivers, along with a password. If the receiver does not expect authenticated messages, then the security segment may be ignored if it is present and the signal may be accepted whether or not it contains a security segment.

FIG. **8C** is a diagram illustrating a security message segment. Security message segment **820** includes a security type field and a security data field **824**. Security type field **822** describes the type of encoding used for security (i.e., MD5, etc.). Security data field **824** contains the data needed to implement the algorithm identified by the security type field **822**.

Detailed Message Descriptions

Wildcard Affinity Update

FIG. **9A** is a diagram illustrating an affinity update wildcard message. Affinity update wildcard message **900** is sent by a service manager to a forwarding agent to register or unregister for classes of flows that match the specified sets of flows. It includes a service message header **902** followed by a sequence of message segments. A security segment **903** is optional, as dictated by the needs of the receiver. A wildcard affinity segment **904** is required, since the purpose of the affinity update wildcard message is to send a wildcard. An action list segment **906** is optional. Its purpose is list the actions that a forwarding agent must support in order to receive the affinity. If the forwarding agent determines that any of the actions are not supported, then it may send an affinity update deny message to the service manager.

An affinity service precedence field **908** is optionally used to specify the precedence of the service being provided. This allows multiple service managers or a single service manager to send wildcard affinities for different services. An affinity backup precedence field **909** is also optionally used to specify the backup precedence of the service manager that sent the affinity. This allows a backup service manager to send wildcard affinities that are ignored until a higher backup service precedence wildcard affinity that corresponds to a primary service manager is deleted. An identity information segment **910** is optionally used to identify the manager. This information may be used, for example, in an error message on the console of the forwarding agent to indicate which service manager had a problem. A service manager interest data segment is optionally used to include data that should be returned to the service manager when an interest match-wildcard affinity message is sent to the service manager as a result of a forwarding agent determining a wildcard affinity match. Finally, one or more action segments are optionally included. The action segments specify actions that are performed on the packets for the purpose of providing a network service. It should be noted that in some embodiments, fields which are described above as optional may become required and required fields may be optional. This is also generally true of the other message descriptions contained herein.

Fixed Affinity Update

FIG. **9B** illustrates a fixed affinity update message that is sent by a service manager to a forwarding agent to add a fixed affinity to the receiver's affinity cache or delete a fixed affinity that is stored in the receiver's affinity cache. If the time to live in the fixed affinity segment is non-zero, the affinity is added to the cache (or refreshed, if it already resides there) for the number of seconds specified in the time to live. If time to live is zero, the fixed affinity is removed from the cache if it is found there.

Fixed affinity update message **920** includes a service message header **922**. An optional security segment **924** is included as dictated by the needs of the receiver. A fixed affinity segment **926** includes the fixed affinity being sent. An affinity service precedence **928** optionally specifies a service precedence. An affinity backup precedence field **929** is also optionally used to specify the backup precedence of the service manager that sent the affinity. This allows a backup service manager to send affinities that are ignored until a higher backup service precedence affinity that corresponds to a primary service manager is deleted. One or more action segments **930** are optionally included to specify actions to be performed by the receiver for matching packets. An identity information segment **932** is optionally used to identify the service manager that sent the fixed affinity. A service manager interest data segment **934** is optionally used to include data that should be returned to the service manager when an interest match-wildcard affinity message is sent to the service manager as a result of a forwarding

agent determining a wildcard affinity match. A forwarding agent interest data segment **936** is optionally used to include data that a forwarding agent requested to be returned to it along with a fixed affinity. Finally, an IP packet segment **938** includes an IP packet.

Usually, the IP packet segment is an IP packet that was sent to a service manager as a result of a wildcard affinity match and that is being sent back to a forwarding agent along with actions to be performed for the packet. In many implementations, the forwarding agent does not devote resources to storing packets that have matched a wildcard affinity and have been forwarded to a service manager. Therefore, the forwarding agent sends the packet to the service manager along with an interest match message and the service manager sends the packet back to the forwarding agent with a fixed affinity update. Thus, the service manager stores the packet for the forwarding agent and returns it to the forwarding agent when the forwarding agent needs to execute an action on the packet. This eliminates the need for storage and garbage collection at the forwarding agent for packets that matched a wildcard affinity and are awaiting instructions from a service manager for handling. In some implementations, the forwarding agents may temporarily store packets that have matched a wildcard affinity. However, it has been found that sending packets to the service manager and having the service manager return packets with fixed affinities simplifies and improves the performance of the forwarding agent.

Affinity Update-deny

FIG. **9C** is a diagram illustrating an affinity update-deny message. An affinity update-deny message is sent by the forwarding agent to a service manager when the forwarding agent receives an affinity update with a required segment that it cannot process (one where the 'Required' flag is set either within the segment header or within the list of segment types from the action list, if one was included). The segments that cannot be processed properly are identified in the action list that is returned with the affinity update-deny message.

Affinity update-deny message **940** includes a service message header **941**. An optional security segment **942** is included as dictated by the needs of the receiver. An action list segment **944** includes actions that are not supported by the forwarding agent and that caused the forwarding agent to sent the affinity update-deny message. An affinity segment **946** from the original affinity update that prompted the affinity update-deny message is optionally included. An identity information segment **948** is from the original affinity update that prompted the affinity update-deny message is also optionally included. A service manager interest data segment **950** is optionally used to include data that the service manager sent to the forwarding agent for the forwarding agent to send back to the service manager when an interest match-wildcard affinity message is sent to the service manager. The service manager interest data is used by the service manager to help process the message. A forwarding agent interest data segment **952** is optionally used to include data that the forwarding agent requests to be returned to it along with a fixed affinity.

Interest Match (Wildcard affinity or Fixed affinity)

FIG. **9D** is a diagram illustrating an interest match message for either a wildcard affinity or a fixed affinity. Interest match message **960** is sent by the forwarding agent to a service manager when an IP packet matches the interest criteria that was sent the last time the matching affinity was refreshed or added in the cache. Interest match message **960** includes a service message header **962**. An optional security segment **964** is included as dictated by the needs of the receiver. An affinity identifier segment **966** includes the affinity key of the affinity that caused the match, the dispatch and information flags of that affinity, and an interest match field that provides reasons from the interest criteria that caused the match. In one embodiment, a bit vector is used to provide the reasons.

An identity information segment **968** is optionally included from the original affinity update that prompted the interest match message to be sent. A service manager interest data segment **970** is optionally used to include data that the service manager requested when an interest match message is sent to the service manager. A forwarding agent interest data segment **972** is optionally used to include data that a forwarding agent requested to be returned to it along with a fixed affinity. Finally, an IP packet segment is optionally included so that the forwarding agent can send the IP packet that caused the affinity match to the service manager. The IP packet is sent if the corresponding data flag in the interest criteria indicated that the IP Packet should be sent. The IP packet may be sent as a segment of the interest match message or may be forwarded independently in a subsequent IP Packet message, depending on the capabilities of the forwarding agent.

IP Packet Only

FIG. **9E** is a diagram illustrating an IP packet only message. IP packet only message **980** is sent by a forwarding agent to a service manager or vice versa whenever an IP network packet is sent from one to the other. This can occur in a number of situations, e.g.:

(1) When a forwarding agent needs to send a service manager a packet that could not be included with an interest match message.

(2) When a forwarding agent needs to send a service manager a packet that matched a service manager wildcard affinity.

(3) When a service manager needs to send a forwarding agent a packet that it has processed and that needs to be forwarded to the next appliance (or, if there are no other appliances, to its correct destination). Encapsulating IP packets in the IP packet only message avoids loops in the system by signaling the forwarding agent that the packet has already been to the manager and need not be sent there again.

IP packet only message **980** includes a service message header **982**. An IP Packet segment **984** includes the IP packet. Preferably IP packet only message **980** does not include a security segment, since the flow is essentially just another IP hop and faster forwarding can be achieved without a security segment.

The messages sent between forwarding agents and service managers have now been described in some detail. The wildcard affinity segment, the fixed affinity segment, and the security segment have also been described. The remaining message segments are described in greater detail below in connection with FIGS. **10A** through **10I**. It should be noted that each segment includes, in addition to the fields that are shown, a segment header.

FIG. **10A** is a diagram illustrating an affinity identifier segment. Affinity identifier segment **1000** includes a dispatch flag **1002**, an information flag **1004**, and an affinity key **1006**. These fields are defined the same as they are defined for fixed affinities and wildcard affinities. Affinity identifier segment **1000** also includes an interest mask **1008** that provides reasons from the interest criteria sent by the service manager that caused the match. This gives the service manager notice of what affinity caused the match and also

what interest criteria in that affinity caused the match. The interest criteria action specified in an affinity sent by a service manager is described further below.

FIG. **10B** is a diagram illustrating an affinity service precedence segment. Affinity service precedence segment **1010** includes a search order flag **1012** that specifies the search order for the precedence, i.e., whether a higher priority precedence is represented by a higher or a lower priority number. A precedence value field **1014** actually provides the precedence value. The service precedence enables one or more service managers to provide different services that are executed in sequential order based on the precedence values provided. In this manner, multiple affinities may be specified that match a flow, with each affinity corresponding to a different service that specifies different actions to be performed for packets in the flow. A packet for such a flow may be forwarded to several service managers before it is eventually sent to the client or the specific server. It should be noted that only the last service manager can dispatch the packet since the packet must be returned by higher priority service managers to the forwarding agent for further processing by lower priority service managers.

Thus, the affinity service precedence allows multiple service managers of different types to control the same flow. The value of the precedence dictates the order in which the forwarding agent should process affinities if multiple matches occur. When a matching affinity contains an action that requires the packet to be sent to a service manager, the action is honored. When the packet is returned, the forwarding agent processes the affinity contained in the response and continues with the matching affinity of the next highest precedence.

FIG. **10C** is a diagram illustrating a service manager interest data segment. Service manager interest data segment **1020** includes an interest data field **1021** that can contain anything that the service manager arbitrarily determines. This is simply data that can be sent by the service manager to the forwarding agent. The forwarding agent returns the data to the manager with an interest match message when an interest match is determined. Typically, this data is used to index the affinity.

FIG. **10D** is a diagram illustrating a forwarding agent interest data segment. Forwarding agent interest data segment **1022** includes an interest data field **1023** that can contain anything that the forwarding agent arbitrarily determines. This is simply data that can be sent by the forwarding agent to the service manager when an interest match is sent to the service manager. The service manager returns the data to the forwarding agent with any fixed affinity update message that is sent as a result of the interest match. Typically, this data is used to index the affinity.

FIG. **10E** is a diagram illustrating an identity information segment that is used to identify the sender of a service message. The identity information may be used for logging and debugging. Identity information segment **1024** includes an IP address field **1025** that contains the IP address of the message sender. A character field **1026** contains the name of the host.

FIG. **10F** is a diagram illustrating a NAT (Network Address Translation) action segment. NAT action segment **1030** includes fields that specify a source IP address **1032**, a source port **1034**, a destination IP address **1036**, and a destination port **1038** that are to replace the corresponding fields in the packet. The NAT action segment thus specifies that NAT is to be performed on any packet that matches the associated affinity. A NAT action segment can be included with any Wildcard or Fixed affinity sent by a service manager to a forwarding agent. The action is not performed on packets that are forwarded to the service manager. If the packet is forwarded to the service manager, then the packet is not immediately altered. If the service manager sends the packet back to the forwarding agent for forwarding, the action is performed by the forwarding agent at that time, therefore removing the need for the manager to implement that function directly.

FIG. **10G** is a diagram illustrating a sequence number adjust action segment. Sequence number adjust action segment **1040** specifies that a forwarding agent should adjust sequence numbers and ACK numbers in the TCP packets that match the associated affinity. A sequence number adjust action segment can be included with any wildcard affinity or fixed affinity sent by a service manager. The sequence number adjust is not performed on packets that are forwarded to the service manager. The action may be performed when the service manager returns the packet back to the forwarding agent for forwarding.

A sequence delta field **1042** specifies the amount by which the sequence number in packets is to be adjusted. An initial sequence number **1044** specifies the lowest sequence number to which the delta is to be applied. An ACK delta field **1046** specifies the amount by which to adjust the ACK number. An Initial ACK number field **1048** specifies the lowest ACK number to which ACK Delta is to be applied. Thus, sequence numbers and ACK numbers in packets can be modified by forwarding agents according to a scheme determined by a service manager. The scheme is sent to the forwarding agents using the sequence number adjust action segment.

FIG. **10H** is a diagram illustrating an advertise action segment. An advertise action segment is sent by a service manager to a forwarding agent to specify that the destination IP address in an enclosed wildcard affinity is to be advertised by the forwarding agent. That means that the address is included in routing protocol updates, just as if the destination IP address belonged to a device connected to the router. The address advertisement is deleted when the associated wildcard affinity is deleted. By directing a forwarding agent to advertise an address, the service manager can simulate the presence of an network service appliance at the location of the forwarding agent. For example, if the service manager is providing load balancing among a group of hosts, the service manager would direct a forwarding agent to advertise the virtual IP address of the cluster of hosts. Thus, the virtual IP address can be advertised as if a load balancer at the location of the forwarding agent were advertising the virtual IP address. If a forwarding agent receives a packet destined for the advertised address, but that packet does not match an affinity (either Full or Wildcard), the packet is dropped. This avoids establishing connections to the forwarding agent for ports that no service manager is supporting.

Advertise action segment **1050** includes an advertise address **1052**, which is the address to be advertised by the forwarding agent. A subnet mask **1054** may also be used for such advertising. If a subnet mask is used, then the IP address and mask combination indicates a subnet to be advertised. The advertise segment can also be used without specifying a subnet mask.

FIG. **10I** is a diagram illustrating an interest criteria action. Interest criteria action **1060** is sent by a service manager to a forwarding agent to specify that the service manager is to be informed when certain types of special packets are detected by the forwarding agent. Interest criteria action **1060** includes an interest IP address **1062** and an interest port **1064**. The interest IP address and port specify

an IP address and port to which the interest match message is to be sent. An interest mask **1066** is bit vector that specifies the types of packets for which the service manager is requesting notification. The type of packet specified by the bits may be a function of the protocol type specified in the affinity encapsulated with the interest criteria action. For example if the protocol is TCP, then in one embodiment, the bits are interpreted as follows:

Bit 0=1:: FIN

Bit 1=1:: SYN

Bit 2=1:: RST

Bit 3=1:: PSH

Bit 4=1:: ACK

Bit 5=1:: URG

Bit 6=1:: Data Present

Bit 7=1:: First Data present

Bit 8=1:: Fragmented packet, and the source/destination IP addresses match

Bit 15=1:: All Packets

If the protocol is UDP, then the bits are interpreted as follows:

Bit 6=1:: Data Present

Bit 7=1:: First Data present

Bit 8=1:: Fragmented packet, and the source/destination IP addresses match

Bit 15=1:: All Packets

For other protocols, Bit **15** may be set to indicate all packets.

A data flag **1067** uses the same bit code as the interest mask. Whereas the interest mask determines whether the service manager should be forwarded an interest match message, data flag **1067** specifies whether the service manager is to receive a copy of the packet that caused the interest match with the interest match message. If a bit is set, then the forwarding agent is to send the packet as well as the interest match to interest IP address **1062** and interest port **1064**. It should be noted that in some embodiments, the forwarding agents may send messages and forward packets to service managers over a different network so that the interest IP address and interest port may not be used or some other method may be used for specifying where interest match messages and packets should be sent to the service manager.

A copy flag **1068** also uses the same bit code as the interest mask. Each bit specifies whether a copy of the matching packet is to be forwarded to the server. If the bit is set for the packet type, the forwarding agent sends a copy of the matching packet and refers to a hold flag **1069** to determine what to do with the original packet. Hold flag **1069** also uses the same bit code as the interest mask. Hold flag **1069** determines whether the forwarding agent forwards the packet to the service manager or, if possible, holds the packet and waits for the service manager to send a fixed affinity that specifies how the packet should be forwarded by the forwarding agent. If the bit is not set for the packet type, then the forwarding agent forwards the packet. If the bit is set, then the forwarding agent holds the packet, if possible. If the packet cannot be held by the forwarding agent for some reason (e.g., lack of storage) then the forwarding agent forwards the packet to the Manager.

FIG. **10J** is a diagram illustrating an action list segment. Action list segment **1070** is sent by a service manager to a forwarding agent with wildcard affinities to specify all the actions that must be supported in order for the forwarding agent accept the wildcard affinity. Action list segment **1070** does not specify that the actions are to be performed. Its purpose is to warn the forwarding agent of the service requirements. The forwarding agent responds with an affinity update-deny and discards a wildcard affinity if the forwarding agent cannot support all the actions in an action list that is provided with the wildcard affinity. Action list segment **1070** includes a first action type **1072**. Action list segment **1070** may also include a second action type **1074** and other action types up to an nth action type **1080**.

A service message protocol for sending messages and packets between service managers and forwarding agents has been defined in FIGS. **6–10J**. Each service message includes a service message header that identifies the message type. After the service message header, each service message includes one or more segments, depending on the message type. Each segment begins with a segment header. Using the message types described, service managers can send forwarding agents instructions detailing certain sets of packets that the service manager wants to either to be forwarded to the service manager or to cause an interest match message to be sent to the service manager. Messages are also used to specify actions for certain packets in certain flows.

For example, if a service manager is providing load balancing, the service manager first sends a wildcard affinity update message to a forwarding agent specifying a set of clients that the service manager will load balance. The wildcard affinity may also include an action that directs the forwarding agent to advertise a virtual IP address for a virtual machine that includes all of the load balanced servers. When the forwarding agent intercepts a packet that matches the wildcard affinity, then the forwarding agent sends an interest match message to the service manager. The service manager then determines a server to assign the connection (or the server that has already been assigned the connection) and sends a fixed affinity to the forwarding agent that directs the forwarding agent to dispatch the packet to that server or to use NAT to substitute the server's address in the packet. The service manager also may include an interest criteria in a fixed affinity that specifies that future packets for the flow should not be sent to the service manager, but that the service manager should be notified if certain types of packets such as a FIN or a FIN ACK are received. At any point, the service manager may cancel a fixed affinity or a wildcard affinity sent to a forwarding agent by sending a fixed affinity or a wildcard affinity with a time to live of 0.

Thus service managers are able to control affinities and monitor flows using the above defined messages. When a forwarding agent receives a packet, affinities received from service managers are searched first for the one with the highest service precedence. Once a match is determined, the search order defined for that precedence is used to find another identical Affinity with a better service precedence. If multiple affinities exist with the same best service precedence, they are searched for the one with the lowest backup precedence value.

Service managers manage the storage of affinities on forwarding agents using the time to live portion of the affinity segments. The forwarding agents remove affinities at intervals specified by the service manager if they have not already been removed at the request of a manager (via an affinity update message with a time-to-live of zero). No affinity is kept for an interval longer than the interval specified by the time-to-live set by the manager (within a tolerance of +/−2 seconds in one embodiment) so that the manager can reliably assume that the affinities have been

cleared at some small time beyond that interval that accounts for any propagation or processing delays. This simplifies the managing of affinities by the service manager across multiple routers. In some cases, a forwarding agent may need to ask for an affinity again if more traffic arrives for that affinity after it has been deleted.

The service manager itself stores affinities long enough to allow forwarding agents sufficient time to delete their own copies. If an affinity is allowed to expire at a service manager, it must be kept by the service manager long enough so that the forwarding agents have deleted their copies first. This avoids mismatches of affinities across routers should a new affinity assignment request be received while a router still has the old affinity.

Service managers also keep affinities long enough after an outbound FIN is detected for a connection so that the final inbound ACK (or in the case of many Windows web browsers, the inbound RST) can be forwarded to the appropriate host. The use of a 'sticky' timer at the service manager satisfies this requirement. If a service manager changes an affinity at a time when it is possible that the affinity is still cached by a forwarding agent, the service manager asks the forwarding agents to delete the affinity before sending the updated affinity.

It should be noted that fixed affinities and wildcard affinities do not themselves include actions in the data structures described above. For flexibility, actions are defined separately but are included with fixed affinities or wildcard affinities in an affinity update message. The associated actions are stored along with the fixed affinity or wildcard affinity on service managers and forwarding agents. Whenever a fixed affinity or a wildcard affinity is referred to as being stored on a forwarding agent or a service manager, it should be understood that associated actions may be stored with the affinity, whether or not such actions are explicitly mentioned.

Likewise, other items may be included in a stored affinity data structure. For example, the affinity may include a time to live when it is sent by a service manager. When the affinity is received by a forwarding agent, the forwarding agent may compute an expiration time from the time to live and store the expiration time along with the fixed affinity.

An architecture that includes service managers and forwarding agents for providing network services has been disclosed. A message protocol for sending messages from service managers to forwarding agents and for reporting activity and forwarding packets from forwarding agents to service managers has been disclosed as well.

Since the service manager does not need to be located at a strategic point in a network, it is possible to include one or more backup service managers at different locations in the network. When a primary service manager fails, a backup service manager can assume the role of the failed primary service manager and provide instructions to forwarding agents so that network services can still be provided.

In order for network services to be provided by a backup service manager without interruption, the backup service manager must receive information about the flows being handled by the primary service manager and store such information so that it is ready to provide instructions to the forwarding agents as soon as the primary service manager fails. Without such state information, the backup service manager would not be able to immediately begin renewing fixed affinities from forwarding agents when those fixed affinities expire. Instead, the newly active backup service manager would need to use its own state machine to determine how forwarded packets from the forwarding agents should be handled. Each connection formerly being handled by the failed service manager would need to be reset and reestablished by the newly active service manager.

In one embodiment, the primary service manager sends a replication packet to the backup service manager whenever a fixed affinity is sent to a forwarding agent. Traffic between the primary service manager and the backup service manager is minimized by having the backup service manager keep track of all such fixed affinities. The fixed affinities expire on the backup service manager in the same manner that the fixed affinities expire on the primary service manager. Thus, messages need not be sent between the primary service manager and the backup service manager for the purpose of timing out fixed affinities. Fixed affinities are timed out by the backup service manager using the same criteria applied by the primary service manager.

Likewise, fixed affinity expiration time intervals are renewed on the backup service manager upon the receipt of a replication packet set by the primary service manager at the same time as a fixed affinity update is sent to a forwarding agent. Thus, the maintenance of fixed affinity on the backup service manager mirrors the maintenance of such fixed affinities on the primary service manager. The backup service manager is thus ready at any point in time to assume the duties of the primary service manager.

FIG. **11** is a block diagram illustrating a distributed network service architecture including service managers and forwarding agents. Forwarding agents **1102***a* and **1102***b* receive packets from networks **1103***a* and **1103***b* and route packets between those networks. A primary service manager **1104** is connected to the forwarding agents for the purpose of providing instructions to the forwarding agents for handling packets and providing network services. Primary service manager, **1104** is in communication with a backup service manager, **1106** via a service manager interface. It should be noted that service manager **1106** may be configured as a backup service manager only or backup service manager **1106** may additionally be configured as a primary service manager providing services for its own set of flows.

Service managers are configured by the system administrator to request certain packets corresponding to certain flows and to specify a backup service priority for the requested flows. Significantly, forwarding agents need not be configured to receive instructions from particular service managers as either primary service managers or backup service managers. The forwarding agent simply follows instructions contained in fixed affinities and wild card affinities received from service managers and prioritizes the instructions according to the backup precedence specified in the affinity update messages received from the service managers.

Both the primary service manager and the backup service manager send wild card affinities to forwarding agents that specify sets of packets to be sent to the service managers. Wild card affinities sent by the backup service manager have a lower backup service priority than the wild card affinities sent by the primary service manager. As a result, forwarding agents forward packets for new flows to the primary service manager. The primary service manager generates fixed affinities for the new flows. The fixed affinities are given a time to live so that they expire on the forwarding agents and forwarding agents must periodically forward packets to the primary service manager so that the fixed affinities may be renewed.

When a packet for a flow is forwarded to the service manager, the service manager resets an expiration time interval for the stored fixed affinity on the primary service

manager and continues to service that flow. The backup service manager likewise stores fixed affinities and allows those fixed affinities to expire after an expiration interval unless a fixed affinity is received from the primary service manager. Thus, the backup service manager maintains a list of fixed affinities that is substantially the same as the list of fixed affinities maintained by the primary service manager. The list is maintained by a combination of receiving replication packets containing the fixed affinities from the primary service manager and allowing the fixed affinities received in the replication packets to expire in a similar manner to the way that the primary service manager allows such fixed affinities to expire.

FIG. **12** is a flow chart illustrating a process executed by a service manager for managing fixed affinities. The process starts at **1202**. In a step **1204**, the service manager receives a message from a forwarding agent. In a step **1206**, the service manager determines if a fixed affinity exists. If the fixed affinity exists, and assuming for the purpose of this example that the service manager does not determine for other reasons that the fixed affinity needs to be changed, the fixed affinity is sent to a forwarding agent in step **1208**. Next, the service manager resets the expiration time of the fixed affinity stored on the service manager. The process ends at **1212**.

If the fixed affinity does not exist, then control is transferred to step **1214** and the service managers state machine generates a fixed affinity. It should also be noted that in certain cases the state machine may determine that an existing fixed affinity may need to be changed in which case the state machine would generate a changed fixed affinity in step **1214**. Control is then transferred to a step **1216** and the service manager forwards the fixed affinity to the forwarding agent. The process then ends at **1218**.

In step **1210**, the service manager resets the expiration time of a stored fixed affinity as a result of a fixed affinity interest match message being received from a forwarding agent. Thus, fixed affinities automatically expire on both forwarding agents and service managers. Forwarding agents have fixed affinities renewed when a service manager resends a new fixed affinity with a time to live specified to the forwarding agent. Service managers renew their fixed affinities when they receive a fixed affinity interest match from a forwarding agent. When a fixed affinity interest match message is received, a service manager resets the expiration time stored along with the fixed affinity.

Thus, fixed affinities are maintained on both the service managers and the forwarding agents on a need-to-know-basis. If a connection terminates in a nonstandard fashion, then the service manager will eventually delete its fixed affinity because no fixed affinity interest match messages will be received from a forwarding agent indicating to the service manager that the connection is active and causing the service manager to reset the expiration time of the corresponding fixed affinity. The forwarding agents will delete their copies of the fixed affinity because they will not receive fixed affinity update messages from the service manager.

Affinities may also be deleted explicitly by a service manager by sending a copy of the fixed affinity to a forwarding agent with a time to live of zero. Fixed affinities are also deleted on a forwarding agent when a wildcard affinity is sent to the forwarding agent that has a time to live of zero and that corresponds to the fixed affinity. In addition, in some embodiments, fixed affinities may automatically be deleted on service managers by deleting associated wildcards. In some embodiments, however, this is not desired as it may be useful to have a service manager continue to handle existing connections until the connections are finished and to cease handling new connections when a wildcard affinity is deleted.

In addition to resetting its own fixed affinity expiration time in step **1210**, the primary service manager also creates a replication packet and sends it to the backup service manager. FIG. **13** is a flowchart illustrating the process for sending a replication packet to the backup service manager. The process starts at **1302** when the fixed affinity expiration time is reset by the primary service manager. In a step **1304**, the primary service manager adds the fixed affinity to a replication packet. In a step **1306**, the replication packet is sent to the backup service manager. The process ends at **1308**.

FIG. **14** is a flowchart illustrating a process implemented on the backup service manager upon the receipt of a replication packet. The process starts at **1402** when the replication packet is received. In a step **1404**, the backup service manager extracts the fixed affinity from the replication packet. Next, in a step **1406**, the backup service manager adds an expiration interval to the current time to derive an expiration time when the fixed affinity stored on the backup service manager will expire. In one embodiment, the expiration interval added on the backup service manager is the same expiration interval used by the primary service manager so that fixed affinities on the backup service manger expire at about the time the corresponding affinities expire on the primary service managers. In some embodiments, the expiration interval on the backup service manager is slightly longer than the expiration interval on the primary service manager to allow extra time for the replication packet to be sent to the backup service manager.

Next in a step **1408**, the backup service manager determines whether a fixed affinity exists. If a fixed affinity does exist, then control is transferred to a step **1410** and the fixed affinity is replaced and the new expiration time is stored. If a fixed affinity does not exist, then control is transferred to a step **1412** and the backup service manager allocates memory and stores the new fixed affinity. Once a fixed affinity has been replaced or a new fixed affinity has been stored, the process ends at **1416**.

It should be noted that the backup service manager may also check to be certain that a wild card affinity exists before storing the fixed affinity received in a replication packet. All fixed affinities should correspond to wild card affinities. The check may be used to detect errors in replication packets received and may also be used in some embodiments to allow different backup service managers to be partitioned to backup different sets of flows for a single primary service manager. The wildcard affinities stored on each of the separate backup service managers partitions the backup service managers. The primary service manager can then broadcast replication packets to all potential backup service mangers since unneeded fixed affinities received in replication packets are not stored by backup service managers that do not also have a corresponding wildcard affinity.

A system for maintaining state information on a backup service manager about flows controlled by a primary service manager has been disclosed. For new flows, the primary service manager sends a replication packet that includes a new fixed affinity to be stored on the backup service manager. When a forwarding agent sends a packet to the primary service manager corresponding to an existing flow, the expiration interval for the corresponding fixed affinity is reset on the primary service manager and another replication packet is sent to the backup service manager so that the corresponding expiration period is reset on the backup

service manager. The backup service manager deletes expired fixed affinities just as the primary service manager deletes expired fixed affinities so that the state of fixed affinities on the backup service manager mirrors the state of fixed affinities on the primary service manager. The primary service manager is not required to notify the backup service manager when fixed affinities are deleted.

It should be noted that when the backup service manager becomes active, the backup service manager may send a message to some or all of the forwarding agents deleting any wildcard or fixed affinities still remaining on the forwarding agents that correspond to the failed primary service manger. The fact that the backup service manager has fixed affinities that correspond to each of the fixed affinities sent by the primary service manager enables the newly active backup service manager to send fixed affinities with immediate expiration times that correspond to all of the fixed affinities stored on forwarding agents that were previously sent by the failed primary service manager.

Although the foregoing invention has been described in some detail for purposes of clarity of understanding, it will be apparent that certain changes and modifications may be practiced within the scope of the appended claims. It should be noted that there are many alternative ways of implementing both the process and apparatus of the present invention. Accordingly, the present embodiments are to be considered as illustrative and not restrictive, and the invention is not to be limited to the details given herein, but may be modified within the scope and equivalents of the appended claims.

What is claimed is:

1. A fault tolerant method of providing a network service, comprising:

receiving a packet corresponding to a flow from a forwarding agent at a primary service manager;

determining at the primary service manager instructions for handling packets corresponding to the flow;

sending the instructions to the forwarding agent;

storing the instructions for handling packets corresponding to the flow at the primary service manager;

sending a replication packet to a backup service manager, the replication packet including the instructions for handling packets corresponding to the flow; and

deleting the instructions for handling packets corresponding to the flow from the primary service manager upon the expiration of a primary service manager instruction maintenance time interval.

2. A fault tolerant method of providing a network service as recited in claim **1** further including storing the instructions for handling packets corresponding to the flow at the backup service manager.

3. A fault tolerant method of providing a network service as recited in claim **1** further including deleting the instructions for handling packets corresponding to the flow from the backup service manager upon the expiration of a backup service manager instruction maintenance time interval.

4. A fault tolerant method of providing a network service as recited in claim **3** wherein the backup service manager instruction maintenance time interval and the primary service manager instruction maintenance time interval are substantially the same.

5. A fault tolerant method of providing a network service as recited in claim **1** further including:

setting a primary service manager expiration time for the instructions for handling packets corresponding to the flow to expire on the primary service manager, the primary service manager expiration time being a primary service manager instruction deletion time interval after the time that the instructions for handling packets corresponding to the flow were stored on the primary service manager; and

setting a backup service manager expiration time for the instructions for handling packets corresponding to the flow to expire on the backup service manager, the backup service manager expiration time being a backup service manager instruction deletion time interval after the time that the instructions for handling packets corresponding to the flow were stored on the backup service manager.

6. A fault tolerant method of providing a network service as recited in claim **5** further including:

receiving a subsequent packet corresponding to the flow from the forwarding agent;

resetting the primary service manager instruction expiration time to the primary service manager instruction deletion time interval after the time that the subsequent packet was received; and

sending a second replication packet to the backup service manager.

7. A fault tolerant method of providing a network service as recited in claim **6** further including receiving the second replication packet at the backup service manager and resetting the backup service manager instruction expiration time to the backup service manager instruction deletion time interval after the time that the second replication packet was received.

8. A fault tolerant method of providing a network service as recited in claim **1** including:

determining at the backup service manager that the primary service manager has failed;

receiving at the backup service manager a subsequent packet from the forwarding agent corresponding to the flow;

matching the subsequent packet to the instructions stored on the backup service manager for handling packets corresponding to the flow; and

sending the instructions stored on the backup service manager for handling packets corresponding to the flow to the forwarding agent.

9. A fault tolerant method of providing a network service as recited in claim **8** further including resetting a backup service manager instruction expiration time to a backup service manager instruction deletion time interval after the time that the subsequent packet was received.

10. A fault tolerant method of providing a network service as recited in claim **8** further including sending canceling instructions from the backup service manager to the forwarding agent upon determining that the primary service manager has failed, the canceling instructions causing instructions from the primary service manager that are stored on the forwarding agent to expire.

11. A primary service manager for providing a network service in a fault tolerant manner, comprising:

a processor configured to determine instructions for handling packets corresponding to a flow;

a forwarding agent interface configured to send the instructions for handling packets to a forwarding agent;

a memory configured to store the instructions for handling packets corresponding to the flow; and

a backup service manager interface configured to send a replication packet to a backup service manager wherein the replication packet includes instructions for handling

packets corresponding to the flow, and wherein the primary service manager is further configured to delete the instructions for handling packets corresponding to the flow upon the expiration of a primary service manager instruction maintenance time interval.

12. A backup service manager for providing a network service in a fault tolerant manner, comprising:

a primary service manager interface configured to receive the instructions for handling packets corresponding to a flow; and

a memory configured to store the instructions for handling packets corresponding to the flow, wherein the primary service manager is further configured to delete the instructions for handling packets corresponding to the flow upon the expiration of a primary service manager instruction maintenance time interval.

13. A fault tolerant distributed system for providing a network service including:

a forwarding agent configured to send a packet corresponding to a flow to a primary service manager;

a primary service manager configured to determine instructions for handling packets corresponding to the flow, to send the instructions for handling packets to the forwarding agent, to store the instructions for handling packets corresponding to the flow and to send a replication packet to a backup service manager, the replication packet including the instructions for handling packets corresponding to the flow; and

a backup service manager configured to receive the instructions for handling packets corresponding to the flow and to store the instructions for handling packets corresponding to the flow, wherein the primary service manager is further configured to delete the instructions for handling packets corresponding to the flow upon the expiration of a primary service manager instruction maintenance time interval.

14. A fault tolerant distributed system as recited in claim 13 wherein the backup service manager is further configured to delete the instructions for handling packets corresponding to the flow upon the expiration of a backup service manager instruction maintenance time interval.

15. A fault tolerant distributed system as recited in claim 14 wherein the backup service manager instruction maintenance time interval and the primary service manager instruction maintenance time interval are substantially the same.

16. A fault tolerant distributed system as recited in claim 13 wherein:

the primary service manager is further configured to set a primary service manager expiration time for the instructions for handling packets corresponding to the flow to expire on the primary service manager, the primary service manager expiration time being a primary service manager instruction deletion time interval after the time that the instructions for handling packets corresponding to the flow were stored on the primary service manager; and

the backup service manager is further configured to set a backup service manager expiration time for the instructions for handling packets corresponding to the flow to expire on the backup service manager, the backup service manager expiration time being a backup service manager instruction deletion time interval after the time that the instructions for handling packets corresponding to the flow were stored on the backup service manager.

17. A fault tolerant distributed system as recited in claim 16 wherein the primary service manager is further configured to receive a subsequent packet corresponding to the flow from the forwarding agent, reset the primary service manager instruction expiration time to the primary service manager instruction deletion time interval after the time that the subsequent packet was received and send a second replication packet to the backup service manager.

18. A fault tolerant distributed system as recited in claim 17 wherein the backup service manager is further configured to receive the second replication packet and reset the backup service manager instruction expiration time to the backup service manager instruction deletion time interval after the time that the second replication packet was received.

19. A fault tolerant distributed system as recited in claim 13 wherein the backup service manager is further configured to determine that the primary service manager has failed, to receive a subsequent packet from the forwarding agent corresponding to the flow, to match the subsequent packet to the instructions stored on the backup service manager for handling packets corresponding to the flow; and to send the instructions stored on the backup service manager for handling packets corresponding to the flow to the forwarding agent.

20. A fault tolerant distributed system as recited in claim 19 wherein the backup service manager is further configured upon determining that the primary service manager has failed to send instructions to the forwarding agent that cause instructions from the primary service manager that are stored on the forwarding agent to expire.

21. A fault tolerant distributed system as recited in claim 19 wherein the backup service manager is further configured to reset the backup service manager instruction expiration time to a backup service manager instruction deletion time interval after the time that the subsequent packet was received.

22. A computer program product for providing a network service in a fault tolerant manner embodied in a computer readable medium comprising computer instructions for:

receiving a packet corresponding to a flow from a forwarding agent at a primary service manager;

determining at the primary service manager instructions for handling packets corresponding to the flow;

sending the instructions to the forwarding agent;

storing the instructions for handling packets corresponding to the flow at the primary service manager; and

sending a replication packet to a backup service manager, the replication packet including the instructions for handling packets corresponding to the flow, wherein the primary service manager is further configured to delete the instructions for handling packets corresponding to the flow upon the expiration of a primary service manager instruction maintenance time interval.

* * * * *